TABLE DES MATIERES

TABLE DES FIGURES

Chapitre I

INTRODUCTION

1.1 Contexte général

La gestion d'un réseau, qu'il soit IPv4 ou IPv6[1], est un sujet très vaste. Il comprend entre autre la gestion du trafic (statistiques, accounting), la gestion des dysfonctionnements, la gestion des configurations, la gestion de l'accès au réseau. Cependant quelque soit le domaine, il nécessite toujours une connaissance exacte de la topologie du réseau. En conséquence, toutes les plates-formes commerciales de supervision de réseaux IPv4 (telles HP OpenView[HPView], ou Tivoli) disposent d'un service de recherche dynamique de topologie.

L'arrivée de réseaux IPv6 natifs amène donc naturellement le besoin d'adaptation des services de recherche dynamique de topologie pour ces réseaux, d'où l'étude de ce qui existait déjà dans ce domaine, pour des réseaux physiquement connectés.

Cette étude, a permis de constater que pour les réseaux IPv6, n'existait qu'un seul outil d'affichage de topologie, et aucun outil de recherche dynamique. De plus, cet outil gère uniquement les backbones disposant de routeurs de type CISCO et utilisant BGP4+[ref] pour protocole de routage. A l'inverse, les outils de recherche dynamique de topologie de réseaux IPv4, gèrent les backbones et les LANs et travaillent en environnement hétérogène.

C'est pourquoi, des travaux de recherches ont été menés pour définir un service de recherche dynamique de topologie pour les réseaux locaux IPv6, en partant des outils existant en IPv4.

Une solution a été retenue et qui a amenée à définir une nouvelle architecture qui est fonctionnelle dans un milieu hétérogène grâce au fait qu'elle se base sur les RFCs[2] et des standards définissant IPv6. Cette architecture se base sur un algorithme distribué de recherche dynamique de topologie.

Notre travail consiste en la conception et l'implémentation d'une infrastructure de visualisation de topologies des réseaux IPv6 se basant sur des algorithmes de collecte de données afin d'assurer une meilleure administration des réseaux et de superviser les éventuelles évolutions de ces derniers.

1.2 Organisation du rapport

Les différents chapitres de ce présent rapport reflètent les étapes d'avancement du projet. Le premier chapitre est un chapitre introductif à travers lequel on va situer le projet dans un cadre bien particulier. Le second chapitre introduit les différents aspects du nouveau protocole IPv6 en montrant ses spécificités et ses apports.

Les deux chapitres qui suivent sont un aperçu des outils existants concernant la visualisation de topologie des réseaux pour les réseaux IPv4 et IPv6 ainsi qu'une petite étude d'une éventuelle portabilité de ces outils et la présentation de la solution retenue.

Ensuite les trois derniers chapitres présenterons la spécification des besoins pour notre travail, la conception proprement dite et enfin sa réalisation : ce qui a été fait et les résultats obtenus. La description de ces différents processus ce fera à l'aide du langage UML.

[1]Internet Protocole version 6
[2]Request For Comments

Enfin nous allons conclure ce rapport par une conclusion générale et une étude des perspectives possibles et envisageables.

1.3 Présentation de l'équipe de recherche :

1.3.1 Historique et organisation du LORIA

Le Laboratoire Lorrain de Recherche en Informatique et ses Applications (LORIA) est une unité mixte de recherche commune au CNRS[3] (Centre National de la Recherche Scientifique), à l'INRIA (Institut National de Recherche en Informatique et en Automatique), à l'INPL (Institut National Polytechnique de Lorraine), et aux universités Henri Poincaré, Nancy 1 et Nancy 2. Cette unité, dont la création a été officialisée le 19 décembre 1997 par la signature du contrat quadriennal avec le Ministère de l'Education Nationale, de la Recherche et de la Technologie et par une convention entre les cinq partenaires, succède ainsi au Centre de Recherche en Informatique de Nancy (CRIN), et aux équipes communes entre celui-ci et l'Unité de Recherche INRIA de Lorraine.

Depuis le 1er janvier 2001, c'est Hélène KIRCHNER qui dirige le LORIA. Actuellement plus de trois cents personnes travaillent dans le laboratoire. Ces personnels sont répartis en 21 équipes de recherche et 7 services d'aide à la recherche. Chaque équipe rassemble des chercheurs, des doctorants et des assistants techniques ou administratifs, pour la réalisation d'un projet de recherche.

1.3.2 Axes de recherches au sein du LORIA

Dans le secteur des sciences et technologies de l'information et de la communication, le LORIA possède, à travers vingt et une équipes de recherche, cent cinquante chercheurs et une centaine de doctorants, des compétences reconnues dans des secteurs en pleine évolution et porteurs de développement économique potentiel. Les activités de ces équipes sont centrées autour de cinq thématiques principales "transversales" sur lesquelles elles développent des recherches fondamentales et appliquées. Bien entendu, des équipes ont des activités dans plusieurs de ces thématiques :

- ✓ Calculs, réseaux et graphismes à hautes performances
- ✓ Télé opérations et assistants intelligents
- ✓ Ingénierie des langues, du document et de l'information scientifique et technique
- ✓ Qualité et sûreté des logiciels et systèmes informatiques
- ✓ Bio informatique et applications à la génomique.

[3] http://www.cnrs.fr/

1.3.3 Présentation de l'équipe MADYNES

MADYNES (**Ma**nagement **Dy**namique **Ne**tworks **S**ervices) est une proposition de projet LORIA INRIA. MADYNES développe des recherches dans le domaine de la supervision et du contrôle des réseaux et des services. Au sein de ce domaine de recherche, MADYNES relève les défis majeurs induits par la dynamicité croissante des infrastructures, protocoles et usages ainsi que par le facteur d'échelle inhérent à l'Internet ubiquitaire.

Les projets de MADYNES sont organisés suivant deux axes orthogonaux. Le premier axe développe des travaux sur l'évolution des modèles génériques et des paradigmes de supervision afin de définir les fondements et l'infrastructure de base pour la gestion autonome. Le second axe contribue à étendre ces fondements au travers des aires fonctionnelles de la supervision. Les aires privilégiées dans le projet sont celles de la sécurité, la configuration de services (de la souscription à l'activation en passant par le déploiement) et le respect des contraintes de performances (paramétrage, monitorage et mesure de qualité).

Ces travaux trouvent de nombreuses applications dans l'Internet nouvelle génération. Outre les couches d'interconnexion, des services à forte dynamicité comme la communication de groupes, les réseaux pair à pair, les intergiciels et les Web services représentent autant de candidats pour la mise en œuvre des concepts conçus dans les deux axes du projet.

MADYNES se positionne dans le thème 1B : Réseaux et Télécommunications de l'INRIA et porte directement sur l'un des cinq axes de recherche du plan stratégique à savoir : Maîtriser l'infrastructure numérique en sachant programmer, calculer et communiquer sur Internet et sur des réseaux hétérogènes. Au niveau local (LORIA), le projet se positionne dans le thème Calculs, Réseaux et Graphismes à hautes performances tout en ayant des interactions fortes avec d'autres thèmes du laboratoire comme : Qualité et Sûreté du Logiciel (QSL) et TOAI (Télé Opérations et Agent Intelligents).

1.3.4 MADYNES et IPv6

L'apparition du nouveau protocole IPv6 a entraîné le besoin de mettre en œuvre des services de gestion spécifiques à ce nouveau protocole. Au contraire de IPv4, les choses ne sont pas aussi abouties dans le monde IPv6 : d'une part, les instrumentations SNMP ne sont pas disponibles, d'autre part, les algorithmes mis en œuvre dans les plates-formes de gestion actuelles sont souvent inutilisables dans un contexte IPv6. L'exemple le plus flagrant est celui des services de découverte de topologie que l'on trouve dans toutes les plates-formes. Tous prennent en entrée des plages d'adresses qu'ils balayent pour découvrir les machines connectées. Cet algorithme n'est pas applicable dans le cas d'IPv6 où les plages d'adresses sont très larges.

Les services spécifiques à IPv6 tels que l'auto-configuration, le multihoming, l'adressage lien-local posent des défis intéressants à la gestion. Une autre phase du déploiement IPv6 est également très attractive pour la recherche en gestion de réseau : c'est la transition. Elle apporte de nombreux services et technologies.

Ces services induisent des besoins de gestion mais également des nouvelles règles de supervision. Par exemple, la protection contre l'utilisation abusive de tunnels v6 requiert le positionnement par gestion des points de terminaison de tunnels sur un réseau.

Dans le cadre du projet MADYNES les travaux sur la supervision des réseaux IPv6 ont débuté en juillet 2001. Les premiers résultats portent sur une architecture hiérarchique d'un service de découverte de topologie. Ces travaux se poursuivent dans le cadre du projet européen IST 6net auquel nous participons. MADYNES participe également à l'élaboration des tests d'implantations d'agents SNMP IPv6.

Un second domaine d'application autour d'IPv6 est celui du test. Dans le contexte de la coopération avec l'Université Columbia de Montréal, des études sur l'utilisation de la technologie active pour les architectures de déploiement de tests de conformité et d'interopérabilité des protocoles de l'Internet ont déjà débuté.

Ces études utilisent l'environnement FLAME développé dans le projet et ses différentes interfaces pour IPv6.

Chapitre II

LE PROTOCOLE IPv6

2.1 Introduction

Une des principales étapes de validation du nouveau protocole Internet IPv6 passe par l'implémentation de protocoles d'administration réseaux supportant les adresses IPv6. Dans ce chapitre, nous allons commencer par présenter le nouveau protocole IPv6. Ensuite, nous allons nous intéresser aux changements introduits à l'interface de programmation des APIs sockets sous UNIX.

2.2 Les apports d'IPv6

La nouveauté majeure d'IPv6 est l'utilisation d'adresses plus longues qu'IPv4. Elles sont codées sur 16 octets et permettent de résoudre le problème qui mit IPv6 à l'ordre du jour : procurer un ensemble d'adresses Internet quasi illimité. IPv4 permet d'adresser 2^{32} = $4{,}29.10^{9}$ adresses tandis que IPv6 permet d'en adresser $2^{128} = 3{,}4.10^{38}$.

La deuxième amélioration importante d'IPv6 est la simplification de l'en-tête des datagrammes. L'en-tête du datagramme de base IPv6 ne comprend que 7 champs (contre 13 pour IPv4). Ce changement permet aux routeurs de traiter les datagrammes plus rapidement et améliore globalement leur débit.

La troisième amélioration consiste à offrir plus de souplesse aux options. Ce changement est essentiel avec le nouvel en-tête, car les champs obligatoires de l'ancienne version sont maintenant devenus optionnels. De plus, la façon dont les options sont représentées est différente, elle permet aux routeurs d'ignorer plus simplement les options qui ne leur sont pas destinées. Cette fonction accélère le temps de traitement des datagrammes. D'autre part, IPv6 apporte une plus grande sécurité : l'authentification et la confidentialité constituent les fonctions de sécurité majeures du protocole IPv6. Finalement, une plus grande attention que par le passé a été accordée aux types de services.

Bien que le champ Type de services du datagramme IPv4 ne soit que très rarement utilisé, la croissance attendue du trafic multimédia dans le futur nécessite de s'y intéresser.

2.3 La notation des adresses IPv6

Une nouvelle notation a été définie pour décrire les adresses IPv6 de 16 octets. Elle comprend 8 groupes de 4 chiffres hexadécimaux séparés avec le symbole deux-points.
Par exemple :

8000:0000:0000:0000:0123:4567:89AB:CDEF

Puisque plusieurs adresses ont de nombreux zéros dans leur libellé, 3 optimisations ont été définies. Tout d'abord, les premiers zéros d'un groupe peuvent être omis, comme par exemple 0123 qui peut s'écrire 123. Ensuite, un ou plusieurs groupes de zéros consécutifs peuvent être remplacés par un double deux-points.
Par exemple :

8000:0000:0000:0000:0123:4567:89AB:CDEF devient 8::123:4567:89AB:CDEF

Enfin, les adresses IPv4 peuvent être écrites en utilisant la représentation de l'adresse en notation décimale pointée précédée d'un double deux-points, comme par exemple :
::192.31.320.46 (adresse IPv4 compatible)
::FFFF:152.81.65.10 (adresse IPv4 mappée)

2.4 Les préfixes

Dire que nous allons utiliser le préfixe FE80::/64 signifie l'utilisation d'adresses ayant les 64 premiers bits qui commencent par FE80 (16 bits pour FE80 et puis 48 bits à zéro). Les 64 bits suivants vont constituer l'identificateur de l'interface. Ainsi, un préfixe s'écrit de la manière suivante : adresse-ipv6 / longueur du préfixe en bits.
Les formes abrégées avec «::» sont autorisées : 8000:3EDC:BA98:7654::/64

Enfin nous pouvons combiner l'adresse d'une interface et la longueur du préfixe réseau associée en une seule notation : 8000:3EDC:BA98:7654:3210:945:1321:ABA8/64

2.5 Les types d'adresses

IPv6 reconnaît trois types d'adresses : unicast, multicast et anycast.

2.5.1 Le type unicast

Une adresse de ce type peut exister sous plusieurs formes :

- Adresse indéterminée. L'adresse indéterminée est utilisée comme adresse source par un équipement du réseau pendant son initialisation, avant d'acquérir une adresse. Sa valeur est 0:0:0:0:0:0:0:0 (ou en abrégé ::).

- Adresse de bouclage (loopback address) Elle vaut 0:0:0:0:0:0:0:1, soit en abrégé ::1.
Il s'agit de l'équivalent de l'adresse 127.0.0.1 d'IPv4.

- Adresses unicast globales : ce type d'adresses est le plus simple. Une adresse globale désigne une interface unique sur l'internet. Un paquet envoyé à une telle adresse sera donc remis à l'interface ainsi identifiée.

- Adresses lien-local : Les adresses de type lien-local sont des adresses dont la portée est restreinte à un lien, c'est-à-dire à l'ensemble des interfaces directement connectées sans routeur intermédiaire : par exemple, des machines branchées sur un même Ethernet, des machines reliées par une connexion PPP[1] ou des extrémités d'un tunnel. Ces adresses sont configurées automatiquement à l'initialisation de l'interface et permettent la communication entre équipements voisins. L'adresse est obtenue en concaténant le préfixe FE80::/64 aux

64 bits de l'identificateur d'interface. Ces adresses sont utilisées par de nombreux dispositifs tels que le protocole de découverte des voisins NDP[2], et de découverte des routeurs RDP[3].

- Adresses site-local : Les adresses site-local sont des adresses dont la portée est restreinte à un site donné. Un site ne possède pas de définition explicite, il est supposé contenir un ensemble de réseaux d'un même emplacement géographique. Ces adresses peuvent être utilisées par les organisations qui ne sont pas encore raccordées à l'Internet afin de ne pas utiliser des adresses IPv6 globales. Une adresse « site-local » est construite en concaténant le préfixe FEC0::/48, un champ de 16 bits permettant de définir plusieurs sous réseaux, et les 64 bits de l'identificateur d'interface. L'utilisation des adresses site-local n'est pas déterminée de façon définitive, en effet des études sont en cours pour décider si ce type d'adresse sera gardé ou non.

2.5.2 Une adresse de type multicast :

Désigne un groupe d'interfaces qui en général appartient à des équipements différents pouvant être situés n'importe où dans l'Internet. Lorsqu'un paquet a pour destination une adresse de type multicast, il est acheminé par le réseau à toutes les interfaces membres de ce groupe. Une adresse multicast est caractérisée par le préfixe FFxy::/16, où x précise si l'adresse est permanente ou temporaire et y définit le « scope ». Le scope indique la portée de l'adresse (niveau de diffusion), ses valeurs possibles sont les suivantes :

0	réservé
1	équipement (node-local scope)
2	lien (link-local scope)
5	site (site-local scope)
8	organisation (organization-local scope)
E	global (global scope)
F	réservé

L'intérêt principal du niveau de diffusion est de permettre de confiner un paquet à l'intérieur d'une zone bien déterminée. Cette méthode est plus rigide mais plus sûre qu'en IPv4, où la portée d'un paquet multicast est définie essentiellement en fonction du champ « durée de vie ».

Il est à noter que les adresses de type broadcast ont disparu du protocole IPv6. La raison de cette disparition est que l'émission d'un paquet broadcast est très pénalisante pour toutes les machines se trouvant sur le même lien. En effet, sur chaque machine le paquet est remonté de la carte réseau jusqu'à la pile IP où il pourra éventuellement être ignoré.

[1]Point-to-Point Protocol
[2]Neighbor Discovery Protocol
[3]Router Discovery Protocol

2.5.3 Adresse anycast

Comme dans le cas du multicast, une adresse de type anycast désigne un groupe d'interfaces, la différence étant que lorsqu'un paquet a pour destination une telle adresse, il est routé à un seul des éléments du groupe et non pas à tous. C'est, par exemple, le plus proche au sens de la métrique des protocoles de routage. Ce type d'adresse est encore en cours d'expérimentation et est réservé pour le moment aux routeurs.

Un paquet IPv6 ne peut en aucun cas avoir pour adresse source une adresse anycast (ou multicast d'ailleurs).

2.6 Portage d'applications vers IPv6

IPv6 résout le problème du nombre d'adresses IP en quadruplant leur taille. Cette augmentation engendre des modifications dans les codes sources des applications. La première étape de notre travail consiste à faire communiquer une entité gestionnaire et une entité agent sur IPv6. Les applications client/serveur, ou plus précisément dans notre contexte gestionnaire/agent, communiquent par l'intermédiaire d'une interface de programmation réseau appelée « **socket** ».

2.7 Interface sockets

Une manière simple d'introduire la notion de socket consiste à faire le parallèle avec la communication de deux individus par courrier [Ste98]. Chacun de ces individus doit disposer d'un point de contact en l'occurrence une boîte aux lettres dans un contexte de communication entre deux ordinateurs. Une socket est donc un point de communication par lequel un processus peut recevoir et émettre des informations.

La création d'une socket se fait par un appel système socket ayant pour paramètres :

- la famille d'adresses AF_INET6 ou PF_INET6 pour TCP et UDP.
- le type de service (SOCK_STREAM pour TCP ou SOCK_DGRAM pour UDP).
- le numéro de protocole à utiliser (IPPROTO_TCP pour TCP et IPPROTO_UDP pour UDP) ou 0 pour utiliser celui par défaut en fonction des deux paramètres précédents.

D'autres familles d'adresses ou de protocoles, ainsi que d'autres types de services sont disponibles, mais n'entrent pas dans le contexte de notre travail.

Puisque nous utilisons le mode de transport UDP, nous allons nous intéresser à la manipulation des sockets avec ce mode de transport non connecté.

Le principe de dialogue entre les deux éléments du réseau se présente comme suit :

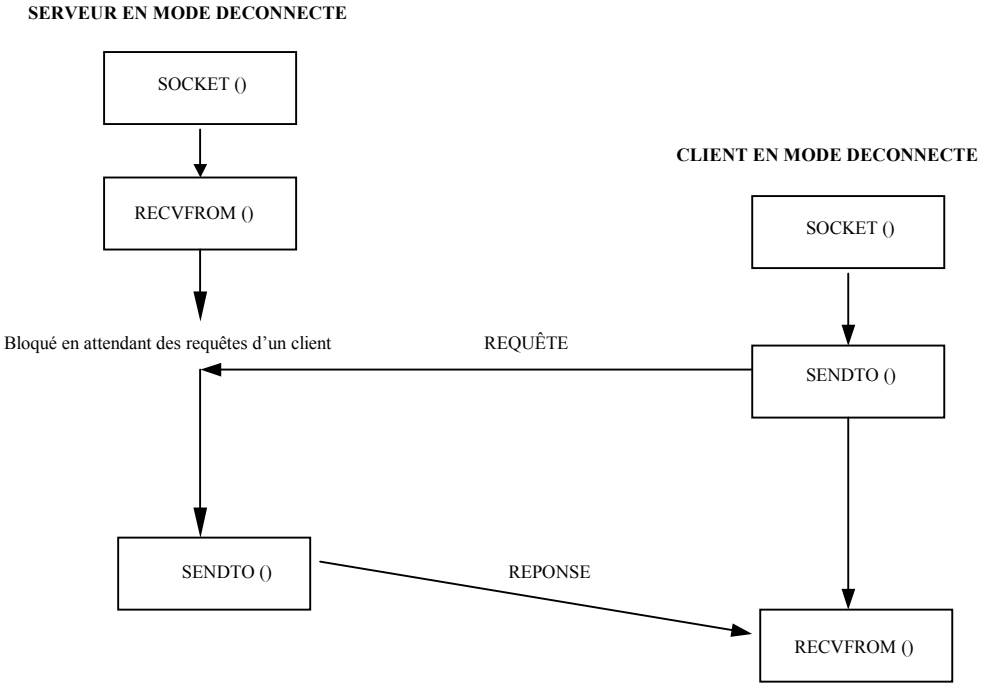


Fig. 1 : Dialogue en mode non connecté

Actuellement, avec l'apparition de la nouvelle pile IPv6, les appels systèmes de l'interface socket (socket, sendto, recvfrom...) n'ont pas changé. Par contre, des modifications ont été introduites au niveau :

- des fonctions du noyau de l'API[4] socket.
- des structures de données dédiées à l'adressage.
- des fonctions de conversion des noms vers les adresses.
- des fonctions de formatage des adresses.

Dans la suite, nous allons présenter les principales modifications introduites à ces fonctions et que nous avons utilisées dans notre implémentation.

2.7.1 Le noyau de l'API socket sous IPv6

La nouvelle API socket a été conçue de manière à introduire le moins de modifications possibles par rapport aux versions IPv4 tout en assurant l'interopérabilité entre les deux versions du protocole.
Les modifications du noyau de l'API socket touchent plus particulièrement les structures de données d'adresses et leurs primitives de conversion. Par exemple, à la création d'une

[4]Application and Programming Interface

socket, il faut spécifier AF_INET6[5] comme famille de socket. Dès lors, les autres fonctions de l'API (bind, connect, sendmsg, sendto, accept, recvfrom, recvmsg) savent à quelle type de socket elles ont affaire et s'adaptent en conséquence. D'autre part, un nouveau numéro pour le protocole IPv6 a été introduit. Dans ce cas, la constante IPPROTO_IPV6 doit être choisie pour décrire une socket qui communique avec la couche IPv6. On peut changer le type d'une socket (fonction setsockopt) grâce à l'option IPV6_ADDFORM. IPV6_UNICAST_HOPS permet quant à elle de limiter la portée des paquets qui sont envoyés via cette socket.

2.7.2 Conversion des noms en adresses

Une nouvelle fonction de résolution de noms a été ajoutée pour prendre en compte les nouvelles et les anciennes adresses :

struct hostent *gethostbyname2 (const char *name, int af).

La valeur de af peut être AF_INET ou AF_INET6. Dans le premier cas cette fonction recherche une adresse IPv4 alors que dans le second elle retourne une adresse IPv6.

La fonction inverse « **gethostbyaddr** » ne change pas puisqu'elle contient déjà un argument af et il suffit de lui donner la valeur AF_INET6 pour qu'elle retourne des noms correspondant à des adresses IPv6.

2.7.3 Fonctions de formatage des adresses

Les fonctions inet_addr et inet_ntoa sont utilisées en IPv4 pour convertir des adresses entre les formes binaires et textes imprimables. IPv6 s'est doté d'outils similaires qui convertissent à la fois des adresses IPv4 et IPv6 :

- ✓ int inet_pton(int af, const char *src, void *dst) : inet_pton convertit une adresse texte vers une forme binaire. La variable **af** spécifie le type d'adresse utilisé IPv4 ou IPv6. **src** est une chaîne de caractères contenant la représentation imprimable de l'adresse à convertir en binaire, et **dst** recevra la forme binaire après conversion. Il faut prendre garde à assurer une taille suffisante à **dst** pour que toute l'adresse puisse être interprétée. Cette fonction retourne 1 si la conversion réussie, 0 si l'adresse n'est pas valide et -1 si **af** est inconnu.

- ✓ const char *inet_ntop(int af, const void *src, char *dst, size_t size) : inet_ntop convertit une forme binaire vers une adresse texte. La variable **af** spécifie le type d'adresse utilisé IPv4 ou IPv6. **src** est un tableau d'octets contenant la représentation binaire de l'adresse à convertir en ascii, **dst** recevra la forme texte après conversion, et **size** contient la taille du tableau d'octets **src**. Cette fonction retourne NULL en cas d'échec et un pointeur sur la chaîne **dst** résultante en cas de succès.

[5]définie dans le fichier /usr/include/netinet/socket.h

2.8 Conclusion

La normalisation du successeur de l'Internet Protocol que nous utilisons tous aujourd'hui est déjà bien avancée, même s'il reste encore beaucoup de travail à faire.

Les premières expérimentations ont déjà commencées, elles sont encore peu nombreuses et le fait de particuliers sans coordination entre eux. Néanmoins de nombreuses implantations sont en tests chez les constructeurs tant pour les postes de travail que pour les équipements de routage. Plusieurs pensent pouvoir fournir une version commerciale dans un délai très rapproché.

Dès lors, la question qui émerge est de savoir qui doit migrer vers IPv6 et quand. La réponse est naturellement liée aux priorités de chacun, mais aussi aux possibilités nouvelles offertes par IPv6 et qui, pour certaines d'entre elles au moins, vont probablement devenir essentielles aux applications courantes de demain : applications de vidéoconférence ou de temps réel, nécessitant une qualité de service garantie, la configuration automatique des équipements, la sécurité des données transportées ...

Chapitre III

ETUDE DES OUTILS EXISTANTS

3.1 Les outils de découverte de topologie pour les réseaux IPv4

Pour les réseaux IPv4, de nombreux outils de découverte dynamique de topologie existent. Presque toutes les plates-formes de supervision fournissent leurs propres outils pour cette fonction. Quelques uns travaillent au niveau des AS (utilisant BGP par exemple), d'autres travaillent à l'intérieur des AS. Ce sont ces derniers que nous allons décrire.

A l'intérieur d'un AS, deux types de services de découverte de réseaux existent :

- Ceux qui construisent les topologies de niveau 3 (appelés aussi outils de niveau 3), déterminant les nœuds et les routeurs du réseau et les connectant avec des routes issues des tables de routage,
- Ceux qui construisent les topologies de niveau 2 (ou outils de niveau 2), découvrant tous les nœuds, routeurs et commutateurs du réseau, et les reliant avec des liens physiques.

La plus part de ces outils utilisent les protocoles ICMP (Internet Control Message Protocol) et SNMP (Simple Network Message Protocol), pour la découverte de la topologie.

3.2 Les outils de découverte de topologie pour les réseaux IPv6

Le seul outil permettant d'afficher une topologie de réseau IPv6 est l'ASPath-tree[1]. Il permet de faire afficher les liens existant entre les routeurs CISCO[2] d'un backbone reliés par le protocole de routage BGP4+. Pour cela, il :

- ✓ Se connecte à chaque routeur du réseau,
- ✓ Récupère les informations concernant les voisins de ce routeur dans la table de routage BGP4+,
- ✓ Recrée les liens entre les différents routeurs afin d'afficher la topologie du backbone.

Cet outil a l'avantage d'être le seul disponible actuellement sur le marché. Cependant, il a été défini pour répondre à un besoin précis, qui était de faire afficher la topologie du 6Bone[3], équipé uniquement de routeurs CISCO. Il ne peut donc pas être utilisé dans sa configuration actuelle pour afficher la topologie d'un backbone hétérogène disposant de routeurs de marques différentes. D'autre part, comme il nécessite la connaissance des logins et passwords d'accès sur chaque routeur géré, il ne peut être utilisé sur un réseau à large échelle et génère un trou de sécurité. Enfin, le besoin de connaissance a priori de ces informations n'en fait certainement pas un outil de recherche dynamique.

3.3 Conclusion

Les services de topologie IPv6 existant ne sont prévus pour fonctionner que dans des contextes homogènes tant du point de vue du type de nœuds (routeurs CISCO) que du type

[1] http://carmen/IPv6/tools/Aspath-ree
[2] http://www.cisco.com

de protocole d'interconnexion entre ces nœuds (BGP 4+). En conséquence, ils ne sont définis que pour traiter des réseaux de type backbone et ne permettent pas à proprement de parler d'offrir une découverte automatique de la topologie de ce type de réseau puisqu'ils nécessitent une liste pré-établie de routeurs à gérer.

Par contre, les services étudiés pour les réseaux IPv4 travaillent tous dans un environnement hétérogène et pourraient être utilisés pour des réseaux de type backbone ou réseaux locaux. Ils offrent pour la plupart une solution de recherche dynamique de la topologie des réseaux, seule la solution distribuée nécessitant l'intervention d'un opérateur pour la récupération des informations stockées dans chacun des serveurs. C'est pourquoi, nous allons réaliser notre étude à partir des services de découverte de topologie associés aux réseaux IPv4.

3.4 Etude de la portabilité des services de IPv4 vers IPv6

3.4.1 Etude de la portabilité des services de niveau réseau

Les services de niveau réseau travaillent selon deux phases : une phase qui peut être nommée recherche exhaustive, qui permet de déterminer l'ensemble des nœuds du réseau géré, et une phase de recherche de squelette du réseau, qui permet de connaître la façon dont ces nœuds sont connectés. Nous allons regarder, phase par phase, si ces différentes parties des algorithmes de recherche de topologie sont utilisables telles quelles pour les réseaux IPv6.

3.4.1.1 Portage de la phase de recherche exhaustive

Elle est basée sur la recherche dans une plage d'adresses de toutes les adresses utilisées. Cette plage d'adresse est définie par la partie identifiant d'interface de l'adresse IPv4. Selon la classe de cette adresse, l'identifiant peut avoir une longueur variant de 8 à 24 bits. Ainsi, pour tester toutes les adresses possibles appartenant à une plage d'adresse, il conviendra de faire entre 2^8 et 2^{24} pings.

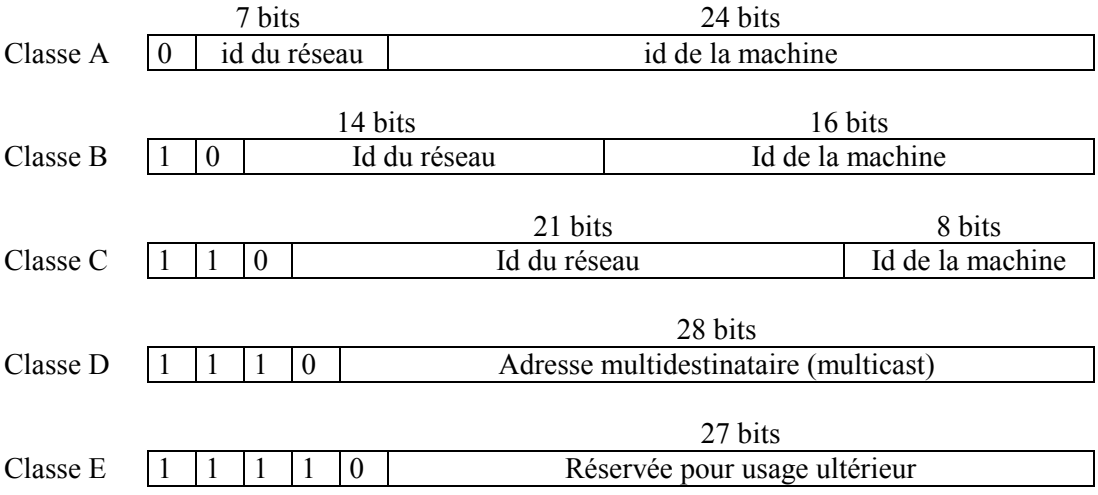


Fig. 2 : Format des adresses IPv4

Mais ce qui était vrai pour IPv4 ne l'est plus pour IPv6. En effet, dans le format d'adressage agrégé, l'identifiant d'interface possède une longueur de 64 bits. En conséquence, pour un réseau IPv6, obtenir la liste des adresses utilisées dans chaque sous réseau revient à faire 2^{64} pings, ce qui est impossible.

3.4.1.2 Portage de la phase de construction du squelette du réseau

L'étude des services de recherche de topologie pour les réseaux IPv4 nous a montré que cette phase pouvait s'effectuer selon deux types d'algorithmes, l'un utilisant SNMP, l'autre utilisant le traceroute.

3.4.1.2.1 La solution SNMP

L'architecture SNMP (RFC 1157 [CFSD90]) est la plus couramment utilisée pour la supervision des réseaux IP. Elle comprend un protocole client/serveur (ou manager/agent dans la terminologie SNMP), le protocole SNMP. L'agent s'exécute sur l'entité à gérer. Il répond aux questions du manager concernant l'état de cette entité. Cet état est fourni par un ensemble d'éléments (ou objets dans la terminologie SNMP) consignés dans une MIB (Management Information Base).

Une MIB est constituée d'objets simples ou structurés (appelés également tables), spécifiées suivant les règles du SMIv2 (Structure of Management Information version 2 [MPS99a] et [MPS99b]). Les tables elles mêmes sont constituées d'objets simples ou de tables. Plusieurs types de MIB existent :

- Celles définies par les RFC qui permettent de gérer tout type d'équipement dialoguant via le protocole IP. La principale est la MIB II qui gère principalement les différents protocoles TCP/IP : IP [McC96a], TCP [McC96b], UDP [McC96c] par exemple. D'autres modélisent le comportement de familles d'équipements (par exemple la MIB des ponts, ou Bridge MIB [DLR+93], etc …).

- Celles définies par les constructeurs et spécifiques à leur matériel, par exemple la MIB CISCO.

Nous ne pouvions envisager de solution reposant sur SNMP, à cause de l'indisponibilité des MIBs, et en particulier de la MIB II (MIB du protocole SNMPv2 supportant IPv6). Cependant, même si elle l'était, la recherche itérative de tous les routeurs du réseau, effectués en demandant l'objet ipRouteNextHop (qui donne l'adresse IP du prochain routeur d'une route) de la table ipRouteTable de la MIB II n'est plus réalisable. En effet, la plupart des protocoles de routage IPv6 existant mémorisent dans l'objet ipRouteNextHop une adresse lien local (cf RFC 2080 [MM97] et RFC 2740 [CFM99]) et non une adresse globale qui permettrait un questionnement au delà du lien sur lequel est connecté l'agent SNMP. Donc, la connaissance de l'adresse du prochain élément ne permet plus de

questionner celui-ci afin de connaître les éléments qui lui sont directement connectés et ainsi, de connaître la topologie globale du réseau. C'est pourquoi le service de recherche dynamique de topologie des réseaux IPv6 n'utilisera pas le contenu de la table de routage.

Ainsi, même si l'architecture SNMP avait été disponible au dessus d'IPv6, seule la solution du traceroute (traceroute6 pour IPv6) était, de toutes façons, envisageable.

3.4.1.2.2 La solution traceroute6

Traceroute6 fonctionne comme le traceroute pour IPv4. Les modifications apportées par IPv6 n'ont donc pas d'influence majeure sur cette fonctionnalité. Cependant, l'utilisation de la fonction traceroute6 pour notre service pose plusieurs problèmes:

- D'une part, à cause de la partialité des informations obtenues. En effet, nous l'avons vu, plusieurs types d'adresses sont définies dans l'architecture d'adressage d'IPv6. Or, le traceroute6 ne nous permettra de connaître qu'un seul type d'adresse pour chaque interface : celui qui sera préféré lors de l'émission.

- D'autre part, le traceroute6 ne nous retourne qu'une seule adresse par nœud. Un nœud comportant plusieurs interfaces ne sera connu que par l'une d'entre elles (cf chemin (1) fig. 3, qui retournera que l'interface I2). La possibilité de faire des traceroute6 dans le sens contraire pourrait permettre de découvrir toutes les interfaces d'un nœud à multiples cartes réseaux (cf chemin (2) fig. 3 qui retournerait I1). Mais se pose alors un troisième problème : comment savoir que ces interfaces, obtenues par des recherches non coordonnées, appartiennent au même nœud ?

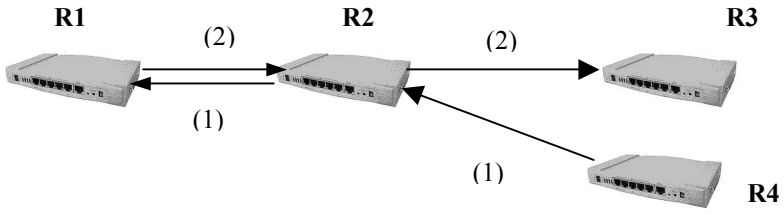


Fig. 3 : Problème de l'association des multiples interfaces

3.4.2 Etude de la portabilité des services de niveau 2

Les services de recherche dynamique de topologie de niveau 2 sont basés à la fois sur des considérations mathématiques et sur le standard SNMP. Si les considérations mathématiques sont indépendantes du protocole IP, nous avons vu qu'il n'en est pas de même pour le standards SNMP. Les services de niveau 2, tels que définis actuellement, ne sont donc pas portables sur IPv6.

Par contre, les services de niveau 3 proposaient des solutions utilisant d'autres protocoles que SNMP. L'étude de leur portabilité sur les réseaux IPv6 est donc justifiée.

3.5 Conclusion

Le portage des services de recherche dynamique de topologie ne peut se faire actuellement que pour les services de niveau 3, le protocole SNMP nécessaire aux outils de niveau 2 n'étant pas encore disponible.

Le portage des outils de Niveau 3 est possible grâce à l'utilisation de la fonction traceroute6, ce qui implique la résolution des quatre problèmes suivants :

- Comment trouver l'ensemble des nœuds du réseau étudié ?
- Comment trouver toutes les adresses d'une interface ?
- Comment trouver toutes les interfaces d'un nœud ?
- Comment associer les interfaces appartenant à un même nœud ?

Chapitre IV

SERVICE DE RECHERCHE DYNAMIQUE DES RESEAUX IPv6

4.1 Architecture proposée

La phase de recherche exhaustive des nœuds du réseau, du chapitre précédent, nous a conduit à implanter, sur chacun des liens, un service qui permet de trouver l'ensemble des interfaces attachées à ce lien. Cependant, le service de recherche dynamique de topologie ne peut pas se limiter à ces éléments puisqu'alors, nous n'aurions qu'une vision locale du réseau, qu'une liste de liens. Il est donc nécessaire de disposer d'un autre outil, capable de relier tous ces liens entre eux. C'est pourquoi nous définissons une architecture hiérarchique à deux niveaux.

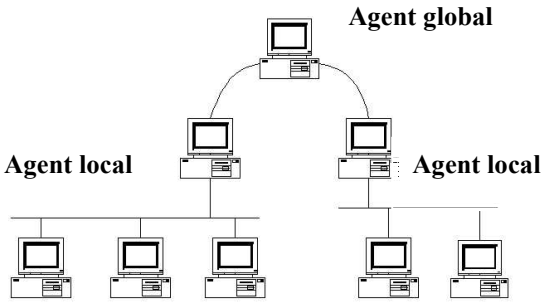


Fig. 4 : Architecture à deux niveaux

Un agent local (AL) installé sur un nœud ne découvre la topologie que d'un seul des liens auquel ce nœud est raccordé. Pour découvrir et gérer tous les liens raccordés à un nœud N, il faut donc implanter autant d'agents locaux que de liens. Ils peuvent être tous implantés sur ce nœud N mais peuvent être aussi implantés sur d'autres nœuds, chacun connecté à un des liens à gérer.

L'architecture comprend donc plusieurs ALs dont la mission est de trouver :

- ✓ L'ensemble des interfaces connectées au lien géré,
- ✓ Pour chacune d'elle :
  - o Son adresse lien local,
  - o Son adresse physique,
  - o Son adresse globale. Le protocole DHCPv6 n'étant qu'à l'état de draft, l'adresse globale de l'interface ne sera cherchée que par reconstruction de l'adresse,
  - o Le type du nœud associé (routeur ou hôte),
  - o Son nom DNS,
  - o Les préfixes utilisés sur le lien.

L' agent global (AG) est unique pour l'ensemble du réseau. Son rôle est de :

- ✓ Recevoir toutes les informations en provenance des agents locaux,
- ✓ Trouver le squelette du réseau,
- ✓ Corréler ces différentes informations,

- ✓ Afficher ou mettre à jour la représentation graphique du réseau.

Les agents locaux doivent connaître une adresse IPv6 de l'agent global. Pour cela, l'agent global se configure avec une adresse de type anycast.

Le protocole utilisé pour le transfert des informations entre les agents locaux et l'agent global peut être un simple protocole au dessus d'UDP car :

- ✓ Les informations sont transmises régulièrement entre l'agent local et l'agent global,
- ✓ L'utilisation de TCP serait trop contraignante : en cas de perte de message, les paquets qui seront retransmis seront strictement identiques à ceux perdus. Or, il peut s'avérer que l'agent local dispose alors d'informations plus récentes qu'il devra attendre pour transmettre.

4.2 Algorithme de l'agent local

L'agent local peut avoir soit un algorithme passif, c'est à dire se contenter d'écouter ce qui se passe sur le réseau, soit un algorithme actif, c'est à dire envoyer des requêtes spécifiques pour obtenir les informations ciblées. Quelque soit le type de phase (passif ou actif), l'algorithme de l'agent local se base sur les messages ICMPv6 de découverte de voisins que nous présentons dans ce qui suit.

4.2.1 Le protocole ICMP

Il fait partie intégrante du protocole IP. Il est définit par le RFC 792 [Pos81], mis à jour par le RFC 950 [MP85]. Il permet de contrôler le fonctionnement d'IP. Avec l'apparition du nouveau protocole IPv6, il fallait réactualiser le protocole de contrôle qui lui est associé. C'est le rôle des RFC 2463 [CD98], 2710 [DFh99] et 2461 [NNS98] qui redéfinissent ICMP pour le protocole IPv6.

L'expérience tirée d'ICMPv4 a permis de mieux spécifier les fonctions de contrôles. Celles ci sont séparées en deux groupes :

- ✓ Le premier contient les messages d'indication d'erreur, renvoyés lorsqu'une erreur est détectée lors de la réception d'une PDU. Ils sont définis dans le RFC 2463,
- ✓ Le second contient : les messages de type information :
  - o Echo Request et Echo Reply, définis également dans le RFC 2463,
  - o Les messages de gestion de groupe multicast, définis dans le RFC 2710,

Les messages de type Echo permettent de tester l'état des nœuds et la connectivité entre deux nœuds. Ils sont donc utilisés à des fins de diagnostic et dans la fonction "ping".

La refonte du protocole ICMP pour les réseaux de type IPv6 a permis d'y intégrer d'autres éléments, comme le protocole de recherche de voisins (RFC 2461), successeur d'ARP pour IPv4.

Ce protocole est utilisé par un nœud, sur un lien, afin de construire son adresse IP et de déterminer le routeur qui pourra router ses paquets vers leur destination. Il comprend 5 messages :

Router solicitation (Solicitation de routeur) : il est émis vers l'adresse multicast regroupant l'ensemble des routeurs d'un lien (FF02::2). Il permet à un équipement d'obtenir des informations de routage, dès le démarrage. L'émetteur peut préciser son adresse physique dans le champ option du message.

Router advertisement (Annonce du routeur) : il peut être une réponse à un message de solicitation mais est aussi émis périodiquement par le routeur. Il contient toujours pour adresse source, l'adresse lien-local du routeur. Dans le cas d'une émission périodique, l'adresse destination est l'adresse multicast définissant tous les nœuds du lien (FF02::1). Deux champs option sont intéressants : celui contenant l'adresse physique du routeur, et celui donnant des informations sur le préfixe (valeur de celui-ci, durée de vie...).

Neighbor solicitation (Solicitation de voisin) : il est utilisé lorsqu'une station émet pour la première fois vers un voisin dont il connaît l'adresse IP et dont il ne connaît pas encore l'adresse physique. Ce message contient l'adresse IP de l'équipement recherché et en général l'adresse physique de la source.

Neighbor advertisement (Annonce du voisin) : il peut être la réponse au message précédent ou être émis régulièrement sur le réseau. Dans le cas d'une émission régulière, l'adresse destination est l'adresse multicast FF02::1. Un drapeau positionné dans l'entête du message permet de déterminer si l'émetteur de ce message est un routeur. Si le message n'est pas une réponse à une solicitation, un champ indique l'adresse IPv6 de l'émetteur. Une option, obligatoire dans le cas d'une réponse à une solicitation, facultative sinon, précise son adresse physique.

Redirect (Indication de redirection) : il est envoyé par un routeur et permet :

- ✓ De fournir à une station, l'adresse du routeur le plus approprié pour faire parvenir son message à destination,
- ✓ D'indiquer qu'une route directe existe entre l'émetteur et le récepteur lorsque les deux stations sont sur un même lien mais disposent de préfixes différents.

Tous les paquets émis par le protocole de recherche de voisins possèdent un TTL de 255.

4.2.2 Fonctionnement de l'agent local

L'avantage d'un algorithme actif est que, bien qu'il augmente le risque de congestion et de collisions sur le lien, il permet d'obtenir à coup sûr les informations ciblées (l'adresse lien local des interfaces et leur adresse physique, par exemple). Á la fin de cette phase active, il envoie l'ensemble des informations recueillies vers l'agent global

A l'inverse, un algorithme passif ne génère pas de trafic supplémentaire sur le lien local. De plus, cette phase permet d'écouter le trafic généré par les protocoles ICMPv6 et de recherche de voisins à savoir les messages de type Neighbor advertisement non sollicités dans lesquels peuvent être inclus l'adresse globale IPv6 de l'interface ayant émis ce message[NNS98]. Mais ces messages ne sont pas des messages périodiques et sont généralement envoyés lorsqu'une interface change de configuration réseau. Donc si la configuration du réseau ne change pas, l'agent local peut attendre longtemps pour obtenir les adresses globales des interfaces connectées au lien.
Cette phase passive lui permet :

- De prendre connaissance le plus rapidement possible d'interfaces nouvellement connectées sur le réseau ou d'interfaces n'ayant pas voulu répondre aux messages ICMPv6 mais générant du trafic. Toute nouvelle information est alors envoyée immédiatement à l'agent global afin qu'il mette à jour sa topologie et la représentation qu'il en a faite.
- De limiter le trafic en n'envoyant que les nouvelles informations reçues et non pas l'ensemble de toutes les informations connues par l'AL, afin d'assurer la mise à jour des informations mémorisées par l'AG, et en particulier, faire disparaître des informations qui ne seraient plus utiles, l'AL recommence une phase active.

C'est la réception des nouveaux vecteurs en provenance des AL qui provoquera la modification de la base de données de l'AG. L'algorithme de l'AL est donc une boucle infinie alternant phases actives et phases passives.

Lors de son initialisation, l'agent local recherche les informations concernant l'interface du nœud sur lequel il se trouve, connectée au lien à gérer. Il effectue ensuite un traceroute6 vers l'extérieur du réseau local et un second vers l'agent global. Ces deux traceroute6 permettent d'obtenir les adresses IPv6 globales des deux routeurs connectés sur le lien géré et se trouvant sur une de ces deux routes. Par l'utilisation d'un Router advertisement, l'agent local peut tenter alors de déterminer leur adresse physique pour pouvoir ultérieurement corréler ces informations avec les informations d'ordre local (adresse lien local et adresse physique) qu'il obtiendra par l'algorithme de recherche exhaustive.

L'algorithme de l'agent local est donc le suivant (les étapes 2 à 8 marquent la phase active tandis que les étapes de 9 à 11 marquent la phase passive) :

1. Recueillir les informations locales au nœud sur lequel l'algorithme est implanté : l'adresse lien local, l'adresse physique, l'adresse globale IPv6 et le nom de l'interface,
2. Envoyer un traceroute6 vers l'extérieur du réseau local et vers l'agent global. Pour chaque routeur trouvé, envoyer un Router solicitation et attendre le Router advertisement associé,
3. Envoyer un ICMPv6 Echo Request vers l'adresse multicast tous les nœuds du lien,
4. Armer une temporisation,
5. Attendre la fin de la temporisation ou l'arrivée d'un message,
6. Si un message arrive :

    - Si le message est un ICMPv6 Echo Reply, envoyer une Neighbor solicitation vers l'adresse source du message reçu,
    - Si le message est un Neighbor advertisement, lire l'adresse IPv6 lien local, l'adresse physique contenue dans le message (s'il y en a une) et le type du nœud (hôte ou routeur). Mettre à jour toute information encore inconnue dans la base de données,

- Si le paquet est un Router advertisement, lire l'adresse IPv6 reçue, l'adresse physique de l'émetteur (s'il y en a une), le préfixe du sous réseau. Mettre à jour toute information encore inconnue dans la base de données,
- Si le paquet est une Neighbor solicitation ou une Router solicitation, lire l'adresse IPv6, l'adresse physique de l'émetteur (si elle est présente). Mettre à jour toute information encore inconnue dans la base de données,

7. Si le temporisateur expire :

   - Si le préfixe est connu, essayer de trouver les adresses globales IPv6 des nœuds ont seules les adresses lien local sont connues, en supposant que les interfaces associées sont des interfaces auto configurées sans état. Envoyer toutes les informations à l'AG. Armer le temporisateur. Aller en 9,
   - Si le préfixe est encore inconnu, armer le temporisateur de nouveau,

8. Aller en 5,
9. Attendre la fin de la temporisation ou l'arrivée d'un message,
10. Si un message arrive :

    - Si le paquet est un Neighbor advertisement, lire l'adresse IPv6, l'adresse physique de l'émetteur (si elle est présente dans le paquet) et son type (routeur ou nœud). Chercher le nom de l'interface si nécessaire. Envoyer toute nouvelle information à l'agent global,
    - Si le paquet est un Router advertisement, lire l'adresse IPv6 et l'adresse physique de l'émetteur, ainsi que le préfixe du sous-réseau. Chercher le nom de l'interface trouvée. Envoyer toute nouvelle information à l'agent global,
    - Si le paquet est une Neighbor solicitation ou une Router solicitation, lire l'adresse IPv6 et l'adresse physique de l'émetteur. Chercher le nom de l'interface associée si nécessaire. Envoyer toute nouvelle information à l'agent global,

11. si le temporisateur expire, aller en 4,
12. aller en 9.

4.3 Algorithme de l'agent global

L'agent global (AG) mémorise toutes les informations en provenance des différents ALs, découvre le « squelette » du réseau et corrèle l'ensemble de ces informations. Son algorithme est le suivant :

1. Découvrir les informations réseaux concernant le nœud sur lequel l'AG est implanté: ses adresses IPv6 globales et lien local, son adresse physique et son nom,
2. Trouver tous les routeurs connectés aux mêmes liens que lui, en envoyant un Router Solicitation vers l'adresse multicast 0xFF02::2. Le Router Solicitation doit donc être envoyé à partir de chaque interface de l'AG,
3. Attendre les messages en provenance de l'AL et des autres routeurs du lien,

4. Rechercher dans les informations déjà reçues si celles que nous venons de recevoir n'ont pas déjà été enregistrées. Si ce n'est pas le cas, les sauvegarder. Il convient ici de vérifier d'une part que des informations ont déjà été reçues en provenance de cet AL, et d'autre part, qu'il n'existe pas de nouvelles données à l'intérieur de ces informations,
5. Sauvegarder l'interface émettrice des informations en provenance de l'AL,
6. Corréler cette information avec celles déjà mémorisées. L'enregistrer si elle n'est pas présente,
7. Effectuer un traceroute6 en direction de l'AL et relier les adresses des routeurs trouvés ainsi avec les informations déjà mémorisées,
8. Mettre à jour la topologie,
9. Aller en 3.

L'étape 5 permet à l'AG de trouver les multiples interfaces d'un nœud car l'interface émettrice de l'AL n'est pas forcément l'interface qui est connectée au lien géré par cet AL.
La corrélation faite à l'étape 6 permet à l'AG d'associer immédiatement les adresses IPv6 trouvées individuellement grâce au traceroute6 avec celles déjà mémorisées grâce aux informations en provenance des ALs. Cela simplifie énormément la construction de la topologie faite à l'étape 8, mise à jour au fur et à mesure de chaque réception d'information fournies par les ALs.

4.4 Implémentation de l'agent global

Nous appellerons vecteur, l'ensemble des informations reçues d'un AL.
Nous appellerons Base de Données des ALs (ou BDLA,) l'ensemble des vecteurs reçus. Elle est constituée d'enregistrements tels que décrits dans la fig. 5).

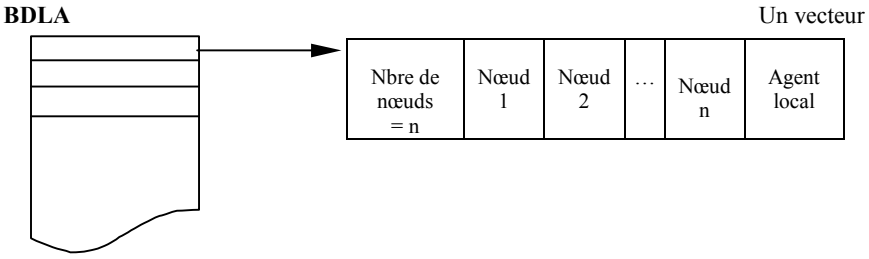


Fig. 5 : Structure de la BDLA

Conjointement à cette BDLA, est créée une base de données des interfaces (ou BI), ensemble des interfaces mémorisées par l'AG. Cette base de données des interfaces est la pièce maîtresse de la recherche dynamique de topologie car elle permet de retrouver rapidement si une interface a déjà été reçue ou pas. Elles est constituée à partir des informations contenues dans les vecteurs mais aussi à partir de celles reçues dans les traceroute6.

La recherche pouvant s'effectuer sur l'adresse IPv6 globale, l'adresse lien local, l'adresse physique ou le nom du nœud, cette BI est en fait composée de quatre listes. Une

table de hachage est associée à chacune des listes afin de faciliter et d'accélérer la recherche des informations.

Ces deux éléments, BDLA et BI, sont nécessaires pour effectuer la corrélation de toutes les informations reçues par l'AG.

Actuellement, la construction de la topologie s'effectue uniquement lors de l'affichage. La structure du réseau local n'est pas mémorisée entièrement. Seules sont mémorisées les relations entre les interfaces d'un même lien (elles appartiennent au même vecteur) et les relations entre les routeurs détectées lors de la réalisation des traceroute6. Ce sont elles qui permettent l'affichage de la topologie complète du réseau.

L'adresse anycast définie pour adresser l'agent global est obtenue à partir du préfixe du réseau géré, fourni lors de la configuration du service de recherche de topologie. Celui-ci est défini par un préfixe particulier, englobant selon le principe de l'adressage agrégé, l'ensemble des sous réseaux le constituant.

Ainsi, un ensemble de sous réseaux définis par les préfixes 2001:660:301:32/64, 2001:660:301:33/64, 2001:660:301:34/64, ..., 2002:660:301:40/64 feront parti du réseau 2001:660:301::/58. L'adresse anycast est obtenue en concaténant le préfixe du réseau avec l'identifiant d'interface de valeur 0x0a. Dans l'exemple ci-dessus, l'adresse anycast de l'agent global opérant sur ce réseau sera donc 2001:660:301::a.

4.5 Envoi des informations entre agent local et agent global

Comme il a été dit précédemment, afin de permettre la réactualisation des données fournies à l'opérateur (perte de connectivité, perte d'un nœud, etc.), les ALs doivent mettre à jour et retransmettre périodiquement leurs informations. Ainsi, le protocole utilisé pour le transfert entre l'AL et l'AG est un simple échange asynchrone de données au-dessus d'UDP. Pour chaque nœud qu'il gère, l'AL utilise une structure (de type nœud) dans laquelle il mémorisera les renseignements obtenus sur ce nœud. La taille de cette structure est un multiple de 64 bits. L'AL les trie selon qu'ils sont routeur, hôte ou de type non encore déterminé (type attendu), afin que l'AG puisse éventuellement faire des traitements spécifiques sur chaque type de nœud.

4.6 Conclusions

L'implémentation faite sur notre plate-forme possède certes encore quelques limites. Mais une évolution de la maquette afin de gérer les multiples préfixes disponibles sur le lien ainsi que le temps de vie des adresses, permettrait de supprimer les limites fonctionnelles. La limite architecturelle liée à la connaissance a priori de la topologie du réseau pourrait être levée par l'utilisation de protocoles actifs permettant de télécharger et d'installer automatiquement les agents locaux sur les liens. De même, les limites liées à l'association des multiples interfaces d'un même noeud disparaîtront lorsque les différents éléments nécessaires seront standardisés. L'utilisation du protocole défini par Matt Crawford [Cra02] permettra d'obtenir, pour toute interface, l'ensemble de ses adresses. La standardisation de la MIB II permettra d'utiliser SNMP et donc d'obtenir les adresses IPv6 globales et lien local de toutes les interfaces d'un noeud.

Chapitre V

SPECIFICATION DES BESOINS

5.1 Etude des besoins de l'application

Le but de notre application est de réaliser un outil de visualisation de topologie (Fig. 6) des réseaux IPv6, disponible sur les environnements Unix FreeBSD. Cette application se base sur une architecture Client/Serveur communiquant à travers des sockets v6 (les sockets utilisant le protocole IPv6). Afin de définir les besoins de l'application, nous devons distinguer trois parties essentielles :

- ✓ La partie serveur qui est une application distribuée consistant à collecter les informations du réseaux et les communiquer à des agents de visualisations
- ✓ La partie client qui est l'agent de visualisation qui doit permettre un affichage clair de la topologie du réseau et offrir certaines fonctionnalités pour l'utilisateur afin d'explorer la topologie du réseau
- ✓ Le protocole d'échange entre clients et serveur qui inclut la phase de connexion et la phase de transfert des données

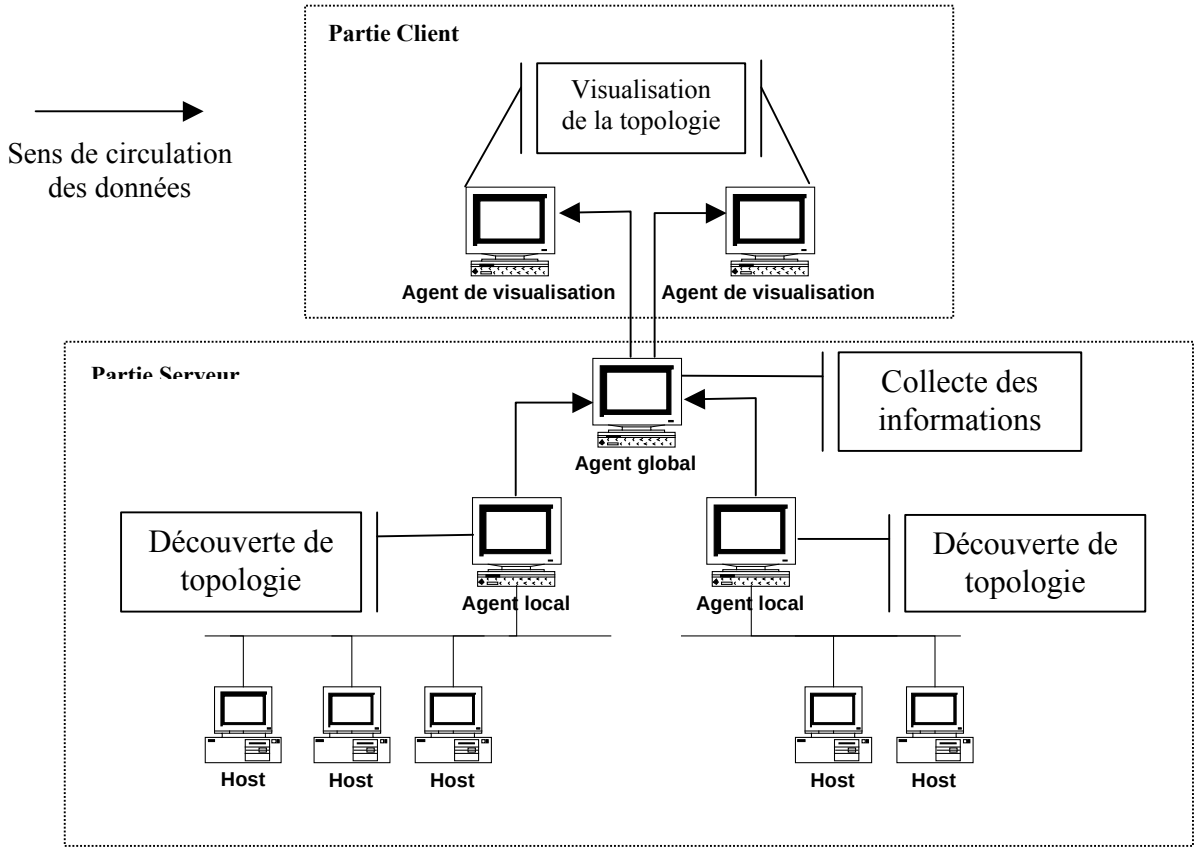


Fig. 6 : Architecture globale de l'outil de découverte

5.2 Spécification de la partie serveur

Le serveur qui fait partie de l'agent global est présenté sous forme de module qui assure les fonctionnalités suivantes :

- ✓ Manipuler les données reçues des agents locaux et les formater suivant le protocole d'échange entre agent global et agents de visualisation,
- ✓ Rester en attente de toute demande de connexion émise par des agents de visualisation, le serveur doit gérer plusieurs connexions simultanées,
- ✓ Envoi des données vers les agents de visualisation,
- ✓ Rester à l'écoute sur le réseau. En cas de modification de la topologie, indiquée par les agents locaux, il doit envoyer la mise à jour vers tous les agents de visualisation,

Toutes ces fonctionnalités seront expliquées en détail à l'aide du langage de spécification des besoins UML à travers les diagrammes de cas d'utilisation et les diagrammes de séquences.

5.2.1 Formatage des données

Cette étape est fondamentale pour le bon déroulement de l'application car les données en question sont celles concernant la topologie et des informations sur les nœuds. L'opération de formatage des données est déclenchée à chaque fois que les informations stockées chez l'agent global sont mises à jours et après corrélation des données provenant des différentes agents locaux. Ce processus est présenté dans ce qui suit à travers les diagrammes des cas d'utilisation et celui d'interactions entre entités y participant :

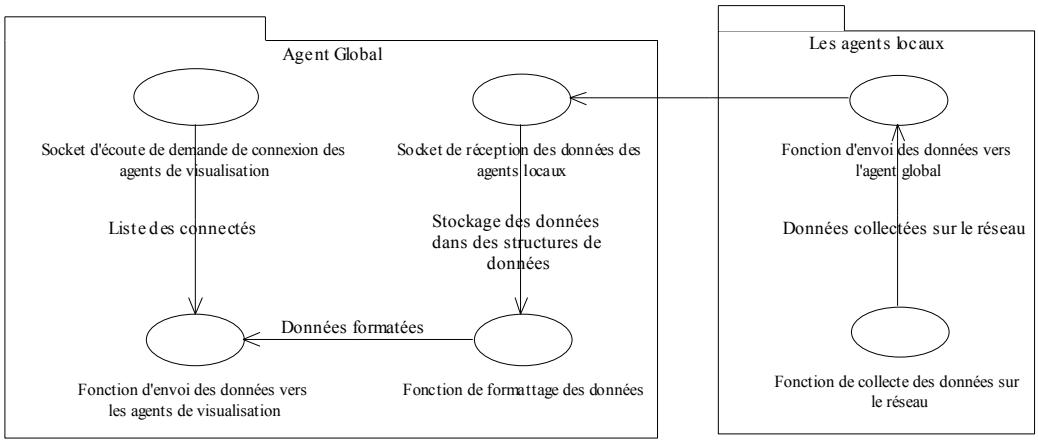


Fig. 7 : Formatage des données : diagrammes de cas d'utilisation

La figure 7 représente le diagramme des cas d'utilisation à travers lequel nous présentons l'invocation du processus de formatage des données.

A travers ce processus nous essayons de structurer les informations concernant la topologie du réseau selon une forme standard pour qu'elles puissent être utilisées par d'autres application utilisant ce standard. En plus ce processus permet la préparation des données pour qu'elles soit envoyées selon le protocole d'échange entre agent de visualisation et agent global. Le standard utilisé pour la représentation des données est XML que nous allons présenter dans la partie conception.

L'interaction entre les différentes entités est présentée par la figure 8 qui explique le déroulement et la succession des différentes actions.

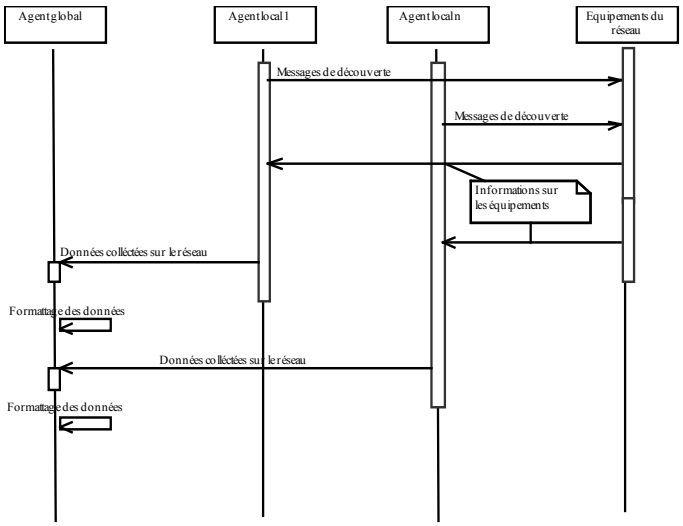


Fig. 8 : Formatage des données : diagramme de séquence

Comme la découverte de topologie se base sur un algorithme distribué, les informations reçues par l'agent global parviennent de plusieurs sources : les agents locaux. Il faut donc corréler les données et gérer les interactions entre elles à chaque fois qu'il y a mise à jour. Ensuite le processus de formatage de données est lancé, ainsi les données seront prêtes à être envoyées vers les agents de visualisation.

5.2.2 Envoie des données vers les agents de visualisation

Cette fonctionnalité est la base de communication entre les agents de visualisation et l'agent global et elle est invoquée dans deux cas :

- ✓ A chaque fois qu'un agent de visualisation se connecte à l'agent global,
- ✓ A chaque fois qu'il y a mise à jour en provenance des agents locaux

Suite à ces événements, l'agent global va envoyer les données qu'il possède et qui ont été formatées précédemment.

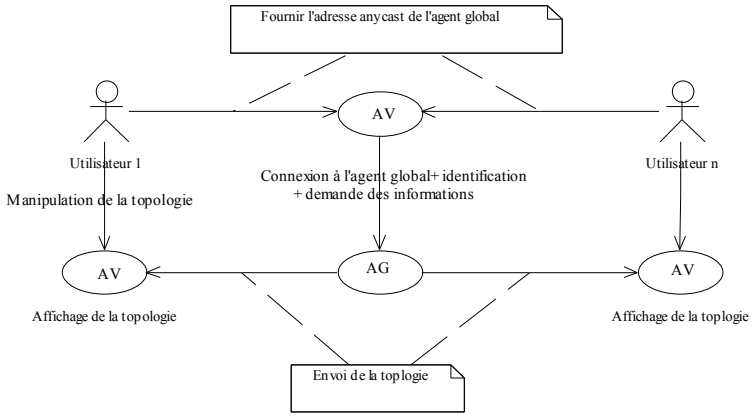


Fig. 9 : Envoie des données : diagrammes de cas d'utilisation

La figure 9 explique l'invocation de l'opération d'envoi de données. En effet, à chaque fois qu'il y a communication entre agent global et les autres agents locaux, qui traduit une mise à jour de la topologie, il y aura automatiquement transfert des nouvelles données vers les agents de visualisation. Il est à signaler que l'agent global est joignable à l'aide d'une adresse anycast, le choix de cette adresse pour se connecter sera expliqué plus en détails dans la partie conception.

En plus de l'adresse anycast l'agent de visualisation doit fournir une clé lors de la connexion pour s'identifier. L'agent global saura qu'il s'agit bien d'un utilisateur qui veut découvrir la topologie du réseau, pourra le gérer et lui envoyer les informations qu'il possède.

L'interaction entre les différentes entités est présentée par le schéma qui suit qui explique le déroulement et la succession des différentes actions.

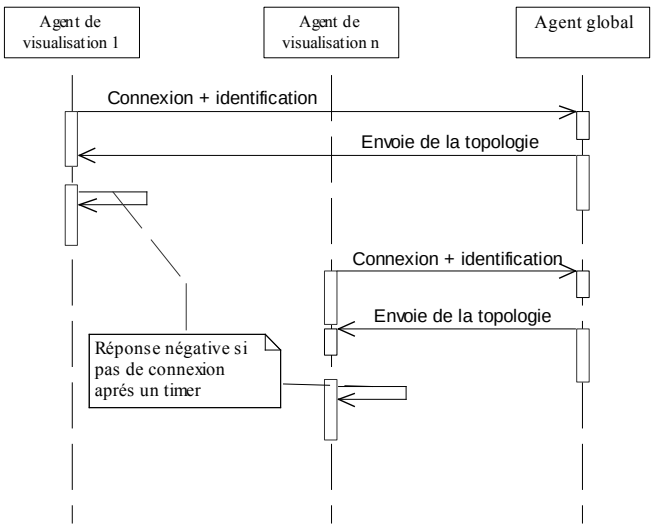


Fig. 10 : Envoie des données : diagrammes de séquences

La figure 10 présente le séquencement des actions lors d'une connexion d'un agent de visualisation. Il finit par l'étape d'envoi des données ou par une erreur dans le cas où il n'y a pas de réponse après l'écoulement d'un timer.

5.2.3 Gestion des rafraîchissements

Cette fonctionnalité est très importante puisque c'est elle qui va assurer la continuité de l'évolution de la topologie au fur et à mesure que les agents locaux la découvrent et qui permet à l'utilisateur de visualiser cette évolution en temps réel.

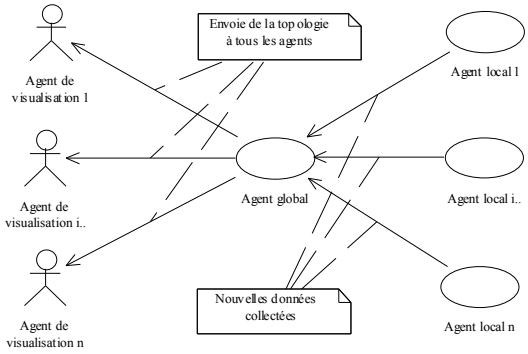


Fig. 11 : Gestion des rafraîchissements : diagramme des cas d'utilisation

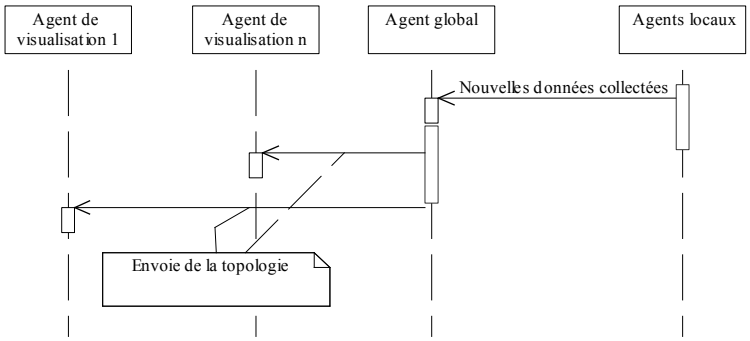


Fig. 12 : Gestion des rafraîchissements : diagramme de séquences

La figure 12 présente le séquencement des actions lors d'une opération de mise à jours invoquée lorsque l'agent global reçoit de nouvelles données de provenance des agents locaux.

5.3 Le protocole d'échange de données

Comme l'application se base sur une architecture Client /Serveur qui se résume au fait que le serveur détient des informations qu'il doit communiquer aux clients afin de les exploiter, cela présume qu'un bon protocole de communication doit être mis en œuvre.

A cette fin nous avons spécifié deux points fondamentaux pour modéliser ce protocole, à savoir le format des trames et le mode d'envoi des paquets entre les deux côtés. Notre choix est le suivant :

- ✓ Utilisation de la technologie XML pour formater les données, à l'intérieur des paquets transmis sur le réseau,
- ✓ Utilisation du protocole UDP comme mode de transmission des paquets

5.4 Spécification de la partie Client « Agent de visualisation »

La partie Client consiste essentiellement en une interface utilisateur, donc le premier objectif à atteindre est de réaliser une interface graphique conviviale et simple à utiliser par tout le monde. Cette interface doit fournir les fonctionnalités suivantes à l'utilisateur de l'application et assurer leur bon fonctionnement :

- ✓ Assurer un affichage clair de la topologie à l'aide d'un algorithme de placement des nœuds et des sous réseaux
- ✓ Afficher les informations concernant chaque nœud du réseau
- ✓ Être indépendante de l'algorithme de recherche
- ✓ Être capable de recevoir des mises à jour à chaque fois qu'il y a modification de la topologie
- ✓ Offrir une bonne maniabilité pour manipuler la topologie

Afin d'assurer ces différentes fonctionnalités l'agent de visualisation doit acquérir les données concernant la topologie depuis l'agent global d'où la nécessité de mettre au point un outil de communication qui permet d'établir des connexions et d'assurer le dialogue entre eux. Pour se connecter à l'agent global, l'agent de visualisation utilise les adresses anycast.

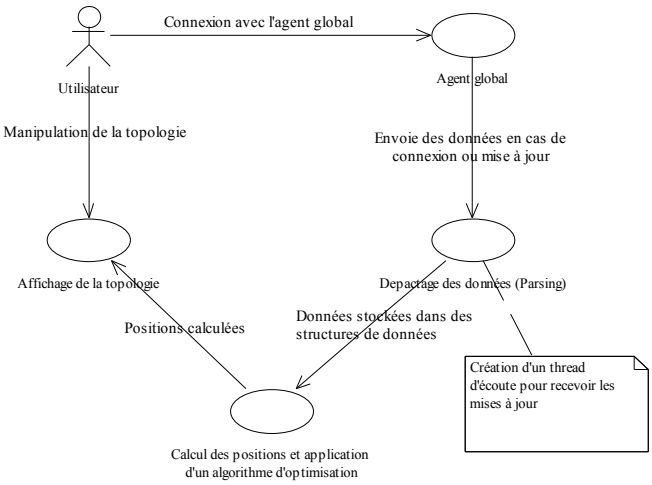


Fig. 13 : Fonctionnement du client : diagramme des cas d'utilisation

La figure 13 explique le fonctionnement du client « agent de visualisation ». En effet lors de sa connexion à l'agent global, ce dernier lui envoie les informations concernant la topologie qu'il possède et qui sont formatées en XML.

Une fois le fichier XML est reçu, il est passé à un analyseur qui va le lire puis va stocker les informations dans des structures de données adéquates. Il passera le résultat à un algorithme dont le rôle est de calculer les positions des différents nœuds et sous réseaux afin d'obtenir un affichage clair. Ensuite un algorithme d'affichage sera lancé pour, enfin, visualiser la topologie à l'utilisateur. Ce dernier pourra la manipuler et demander des informations concernant des nœuds par des simples clics de la souris.

Lors du déroulement du processus d'affichage et la gestion des interactions entre interface et utilisateur, un autre processus reste à l'écoute sur le réseau à travers une socket connectée à l'agent global et qui permet de recevoir des rafraîchissements s'ils ont lieu entre temps et relancer ainsi tout le processus. L'utilisateur pourra ainsi suivre l'évolution de la topologie du réseau en temps réel.

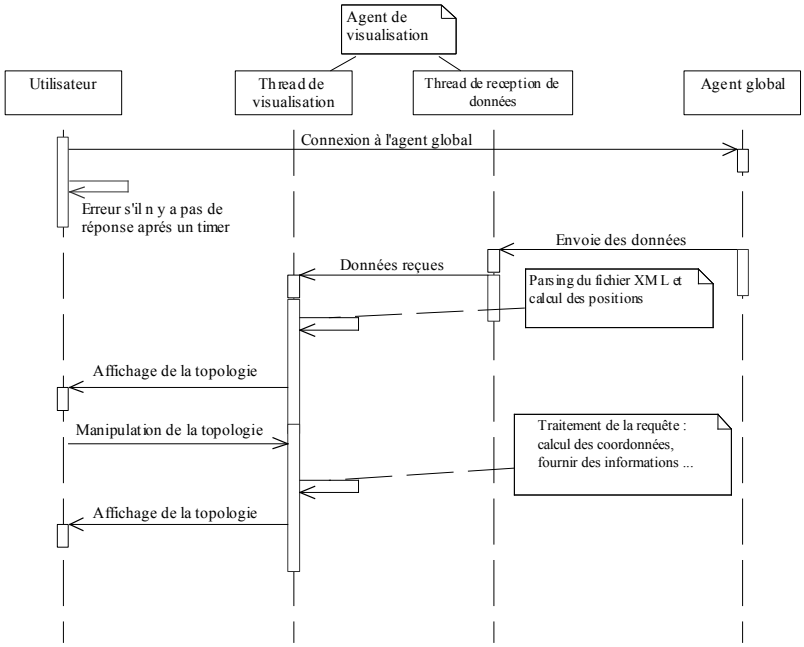


Fig. 14 : Fonctionnement du client : diagramme de séquences

La figure 14 présente le séquencement des actions qui traduisent le fonctionnement du client « agent de visualisation ».

Chapitre VI

LA CONCEPTION

6.1 Conception générale de l'application :

La conception de logiciels fait sans aucun doute appel à l'art dans une large mesure. Mais, en l'absence de règles, l'expression artistique se transforme en une conception chaotique. Pour produire un système ouvert, nous devons utiliser quelques règles bien définies pour gouverner les interactions entre les systèmes et les sous-systèmes .

Ce chapitre est réservé pour la conception des modules du travail selon la méthodologie OMT. La notation UML sera par ailleurs utilisée pour la conception des différents modules vu l'intérêt qu'elle présente.

La conception de notre application, qui se base sur une architecture Client/Serveur, a du passer par plusieurs étapes qui se divisent en trois grandes parties à savoir la conception de la partie serveur (agent global), celle des clients (agents de visualisation) et enfin du protocole de transfert de données. A présent nous allons expliquer plus en détails ces différentes parties.

6.2 Conception de la partie serveur

Cette partie qui est incluse dans l'agent global doit assurer différentes fonctionnalités à savoir la gestion des connexions des agents locaux et des agents de visualisations et la gestion des données que ce soit celle qui sont reçues ou celles qu'il va envoyer. Pour assurer ces fonctionnalités nous avons utilisé les sockets v6 comme moyen de communication et la technologie XML pour le formatage des données.

Comme l'agent global est écrit en langage C, nous avons du suivre l'esprit de sa conception et le compléter à fin de l'adapter pour y intégrer nos nouvelles fonctionnalités. Les différentes procédures et fonctions utilisées dans cette partie sont distribuées dans des fichiers « .c » et des fichiers d'entête « .h ». Les interactions entre ces fichiers se résument en des appels de fonctions ou l'utilisation des structures de données et des variables globales.

Dans ce qui suit nous allons présenter sous forme de schéma la plupart des fichiers et les fonctions qu'ils présentent et les interactions qui existent entre ces différents fichiers. Les fonctions et les interactions ne seront pas toutes présentées pour ne pas encombrer le schéma, mais nous allons montrer les plus importantes vis-à-vis de l'objectif que doit assurer cette partie. Elles seront détaillées après.

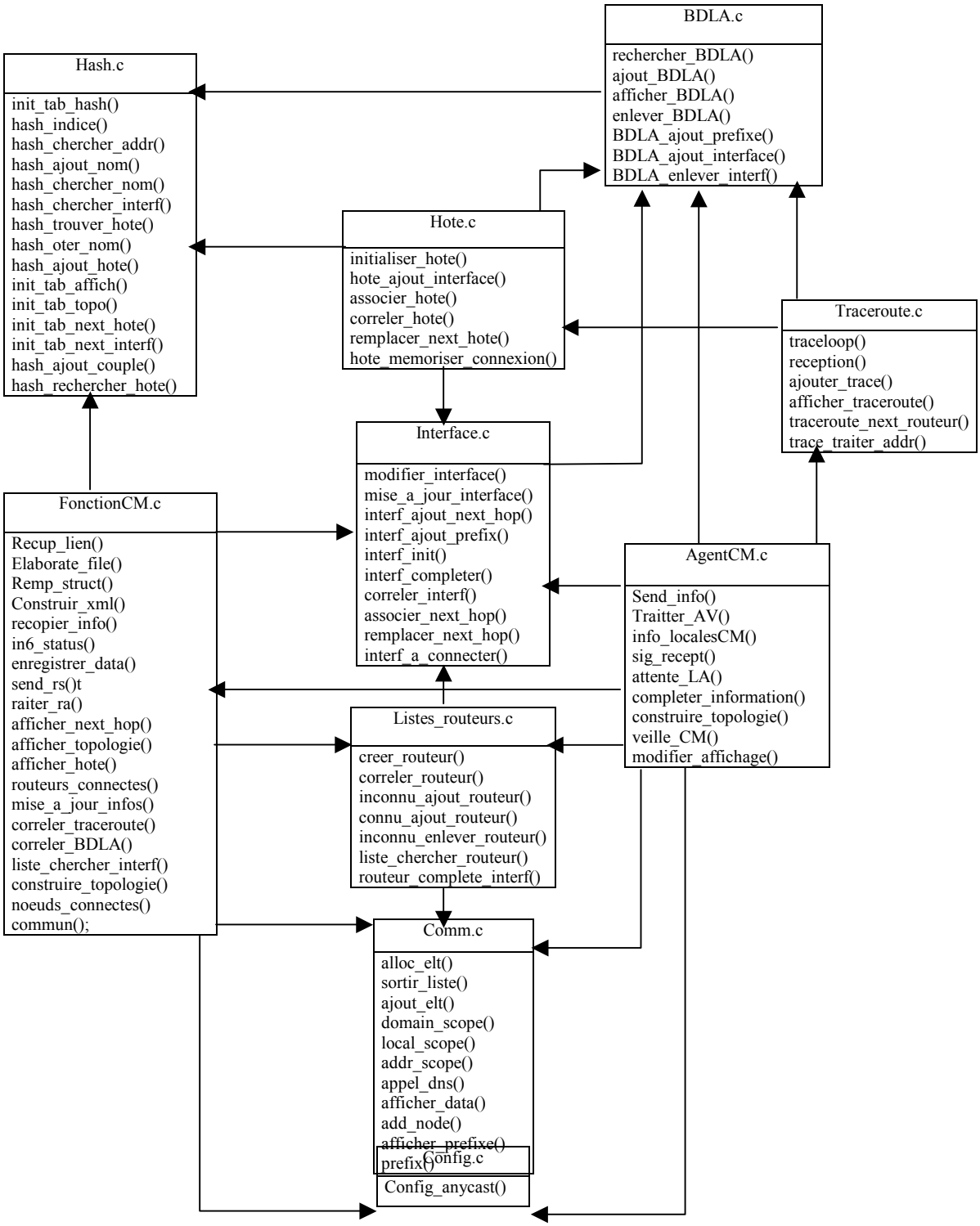


Figure 15 : Répartition des tâches pour la partie serveur

Lors de son lancement, l'agent global va exécuter un certain nombre de procédures et de fonctions, les plus importantes sont :

La première fonction à exécuter est la fonction main() du fichier agentCM.c. Celle-ci fait appel à la fonction info_locales() qui va récolter les informations du nœud sur lequel se trouve l'agent global. Ensuite l'agent global va se mettre en attente d'éventuelles connexions des agents de visualisations ou des informations de provenance des agents locaux, grâce à la procédure attente_LA().

Cette dernière procédure utilise les sockets v6 pour rester en écoute sur le réseau. Pour différencier les messages en provenance d'un agent de visualisation ou d'un agent local, cette fonction étudie l'entête de la trame.

Lors de la réception de données de provenance d'un agent local, la fonction enregistrer_data() du fichier fonctionCM.c sera exécutée. Cette dernière a pour rôle d'enregistrer les nouvelles données et de les fusionner avec les anciennes. Pour cela, elle fait appel à différentes fonctions comme les fonctions de hachage du fichier hach.c et des fonctions de corrélation de données du fichier BDLA.c qui permettent de mettre à jours les BDLA (base de données des agents locaux).

Une fois les données corrélées, l'agent global met les informations de topologie dans les structures de données correspondantes à l'aide des fonctions suivantes :

- ✓ La fonction Remp_struct() qui est une fonction récursive qui collecte les données des différentes BDLA corrélées et les stocke en tenant des résultats des traceroutes effectués par l'agent global,
- ✓ La fonction add_node(), est appelée par la fonction remp_struct(). Elle permet d'ajouter un nouveau nœud et les informations qui lui sont associées, à savoir les interfaces qu'il possède, les liens sur lesquels il se trouve, son nom, les adresses physiques, globales, lien local et les noms de ses interfaces,
- ✓ Gest_node_id() et gest_link_id() : ces deux fonctions gèrent l'attribution des identificateurs pour les nœuds et les liens,
- ✓ La fonction recup_lien() : cette fonction est aussi appelée lors du stockage des données, elle permet de récupérer les informations des liens : les interfaces et le nombre de nœuds du lien,

Ensuite l'agent global procède au formatage des données en XML et la création du fichier XML. Il fait donc appel aux fonctions construir_XML et elaborate_file().

Pour comprendre la technologie XML et son utilité, nous allons la décrire et montrer sa spécificité dans ce qui suit.

6.2.1 La technologie XML

6.2.1.1 Définition

XML (E**X**tensible **M**arkup **L**anguage)est un langage de codage de données dont l'objectif est, dans un échange entre systèmes informatiques, de transférer, en même temps, des données et leurs structures.

Permettant de coder n'importe quel type de donnée, depuis l'échange EDI (**E**lectronic **D**ata **I**nterchange) jusqu'aux documents les plus complexes en passant par les échanges de données inter-applications, son potentiel est de devenir le standard universel et multilingue d'échange d'informations. Appliqué aux documents textuels, il permet d'identifier, de façon logique, la structure et l'organisation de l'information textuelle.

6.2.1.2 Objectifs

XML est un format textuel très flexible dérivé de SGML (**S**tandard **G**eneralized **M**arkup **L**anguage). Initialement conçu pour relever les défis de l'édition électronique de grande puissance, XML joue également un rôle de plus en plus important dans l'échange d'une grande variété de données, que ce soit sur le Web ou pour n'importe quel échange inter-applicatif.

XML permettra, comme le souligne le W3C :

- ✓ L'édition électronique internationalisée, de façon indépendante des logiciels et des systèmes.
- ✓ Aux industries de définir des protocoles, indépendants des logiciels et des systèmes, pour l'échange des données (particulièrement les données du commerce électronique).
- ✓ De fournir de l'information aux agents utilisateurs sous une forme qui permette un traitement automatique après réception, par exemple pour toutes les applications de téléphonie mobile.
- ✓ De faciliter le développement logiciel dès lors qu'il s'agit de manipuler l'information spécialisée et répartie.
- ✓ De faciliter les traitements de données avec des logiciels peu coûteux, à ce titre l'avenir d'XML et d'un certain nombre de ses recommandations associées (XSLT, XPath, Infoset, etc.) est de devenir partie intégrante des couches hautes des systèmes d'exploitation.
- ✓ Aux utilisateurs du Web d'afficher l'information reçue avec la feuille de styles qu'ils souhaitent.
- ✓ De faciliter la fourniture de Méta données (données descriptives de documents) qui aide à trouver de l'information.

6.2.1.3 Principes

Un document XML se compose, d'une part, de texte, et d'autre part, d'informations de structure. Les informations de structure servent le plus souvent à délimiter du texte, pour en identifier la sémantique. Ainsi, <métier>Consultant</métier> permet de dire que la chaîne de caractères "Consultant" doit être comprise comme étant une définition de métier. Il est possible de délimiter des chaînes de caractères. Il est aussi possible de délimiter tout ensemble d'informations mélangeant texte et structure. Par exemple, dans le document suivant :

```
<métier>
        <nom>
                Consultant
        </nom>
        <descr>
                Le consultant travaille pour des consultés qui ...
        </descr>
</métier>
```

La notion de métier introduit, d'une part, un nom de métier et, d'autre part, une description de métier. C'est l'appartenance hiérarchique qui définit que tout cela parle bien du même métier : elle permet de spécialiser des descriptions : un nom de métier et une description de métier.

Pour compléter cette description, il est parfois nécessaire de *valuer* la signification d'un objet. Par exemple, <auteur affiliation="rennes2">Jean Dupont</auteur>, permet d'identifier une chaîne de caractères comme étant un auteur et, en plus, de décrire cet auteur, par *valuation*, comme appartenant à l'Université de Rennes II.

Les balises (<ma Balise> ... </ma Balise>) délimitent des objets typés. Les attributs (<ma Balise type="standard">) définissent des *valuations* d'objets. Un document XML est alors un arbre d'objets typés et *valués*.

Pour finir, un document XML doit définir le jeu de caractères qu'il utilise, ainsi que la version de la recommandation XML. L'exemple précédent s'écrira alors :

```
<?xml version="1.0" encoding="utf-8"?>
<métier>
    ...
</métier>
```

Ce document est parfaitement décrit, il représente, au sens de la recommandation, un document "bien formé" (*wellformed*). Si nécessaire, un modèle documentaire peut lui être adjoint. Celui-ci définira les contraintes associées à ce document : le document devra alors être "valide", au regard de ce modèle. Définis de façon électronique, les applicatifs de type *parser*s seront capables de valider, de façon automatisée, la conformité d'un document à sa classe, à son modèle. L'avantage ? Il sera possible d'appliquer des processus automatiques

sur une classe de documents (les algorithmes étant écrits au regard de la classe et non pas au regard de chaque instance de document).

6.2.1.4 Les différents aspects de la recommandation XML

La recommandation XML s'intéresse à des notions fort différentes :

- ✓ Un langage de codage, basé sur Unicode XML, de documents, avec des éléments, des attributs, un jeu de caractères, etc.
- ✓ Un langage de définition de modèles documentaires, avec les DTD (Document Type Definition).
- ✓ Un langage d'expression d'inclusions de fichiers, avec les `internal` et `external subset` des DTD.
- ✓ L'expression d'informations applicatives, comme les deux attributs `xml:lang` et `xml:space`, dont le rôle est, respectivement, d'indiquer la langue de rédaction d'un fragment de contenu et la façon de traiter les caractères d'espacement d'un contenu.
- ✓ Un langage de prise en compte extrêmement limité des espaces de noms.

Le fait de mettre tous ces aspects dans une même recommandation a un aspect politique important, car cela oblige à prendre *tout* en compte sans différenciation. D'un point de vue technique, cela peut parfois poser des problèmes de compréhension : par exemple, pourquoi prendre en compte seulement les notions de langue et pas celles d'URI (**U**niform **R**esource **I**dentifiers)? Cela peut aussi poser des problèmes d'architecture d'application, dès lors que, par exemple, un Schéma permet d'exprimer la même chose, et davantage qu'une DTD... sauf les notions d'inclusions de données spécifiées de façon indépendante des documents eux-mêmes, au travers des entités générales.

6.2.2 Format du fichier

Vue l'importance des avantages que présente la technologie XML dans le domaine de codage des données et le fait qu'elle soit un standard pour la représentation des données, nous l'avons choisit pour l'adapter à notre application.

La figure 16 illustre la disposition du fichier XML telle que nous l'avons spécifié.

Fig. 16 : Spécification du fichier XML

Une fois le fichier XML prêt, l'agent global consulte une liste dans laquelle se trouvent les adresses des agents de visualisation connectés et leurs envoie le fichier XML dans lequel se trouvent toutes les informations de la topologie du réseau. Il utilise pour cela les fonctions traiter_AV() et send_info(). Ces deux fonctions sont aussi appelées lors de la demande de connexion d'un agent de visualisation ou lors d'une mise à jour de la topologie, elles permettent d'envoyer les données concernant la topologie.

6.3 Conception de l'agent de visualisation

L'agent de visualisation permet de se connecter à l'agent global pour acquérir les informations qui y sont stockées concernant la topologie du réseau. Ensuite, il affiche la topologie du réseau et permet à l'utilisateur de la manipuler et demander des informations concernant des équipements du réseau.

Dans ce qui suit, nous allons détailler la conception de cette partie en explicitant les différents objets crées et leurs principales méthodes, la notation UML sera par ailleurs utilisée.

6.3.1 conception générale

La conception de l'agent de visualisation repose sur un modèle objet à travers lequel nous avons essayé de ramener les objets du monde réel en objets abstraits grâce au langage de programmation orienté objets. Dans ce qui suit nous allons nous intéresser à expliciter les différentes classes élaborées durant le cycle de développement de l'application.

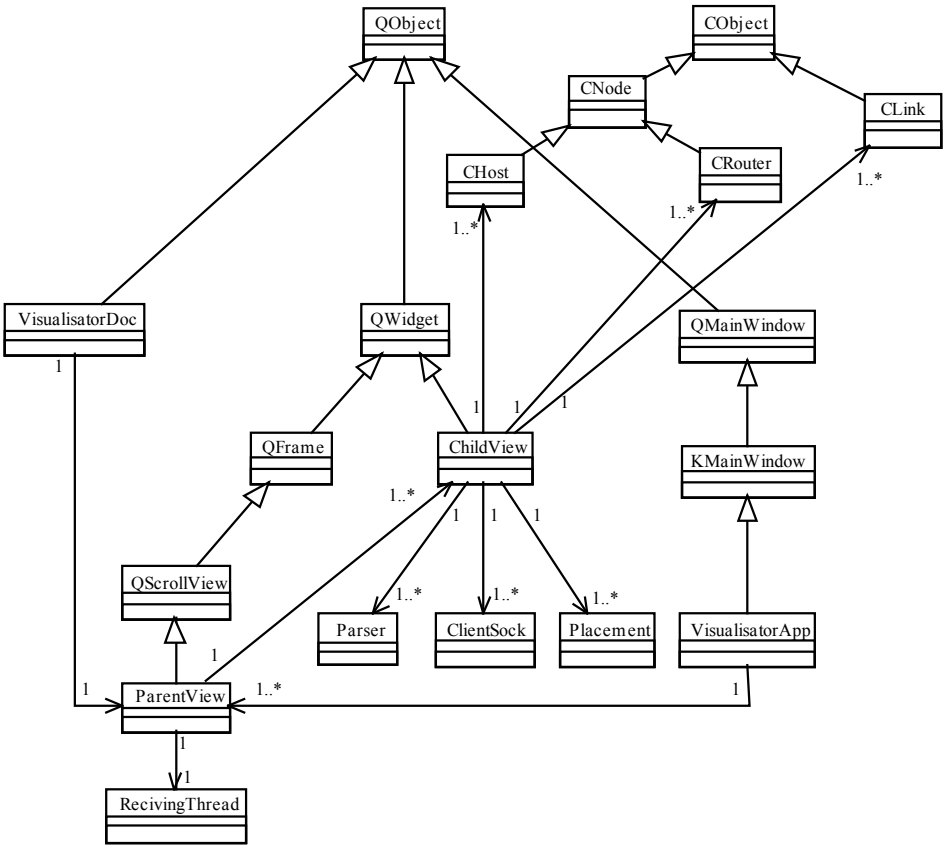


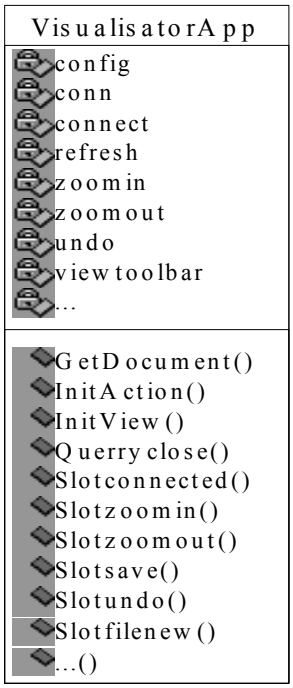
Fig. 17 : Diagramme des classes

La figure 17 présente les principales classes utilisées lors de l'élaboration de ce travail. Le schéma présente aussi les différentes interactions existantes entres les classes à savoir les relations d'héritage, d'instanciation et de généralisation.

Pour mieux comprendre le fonctionnement de ces classes nous allons détailler les plus importantes d'entre elles dans la partie conception détaillée.

6.3.2 Conception détaillée

6.3.2.1 La classe VisualisatorApp

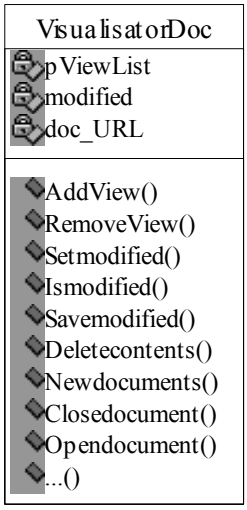


La classe VisualisatorApp est la classe principale de l'application car elle gère les différentes vues (View) et documents (Doc) de l'application. En effet tous les signaux et les actions externes et internes passent par cette classe qui les transmet vers les méthodes des autres classes à fin de les traiter. Cette classe permet aussi d'instancier d'autres classes et de les initialiser comme la classe ParentView et VisualisatorDoc.

La plupart des méthodes de cette classe servent à traiter les différents signaux et actions transmis lorsque l'utilisateur manipule l'application, ces actions peuvent être des demandes de connexion qui sont assurées par la fonction SlotConnected(), des opérations de zoom assurées par les méthodes SlotZoomIn() et SlotZoomOut(), de fermetures de l'application ou de la connexion assurées par les méthodes QueryClose() et QueryExit(), des opérations d'enregistrement de la topologie, d'ouverture de fichiers…

Fig. 18 : La classe VisualisatorApp

6.3.2.2 La classe VisualisatorDoc



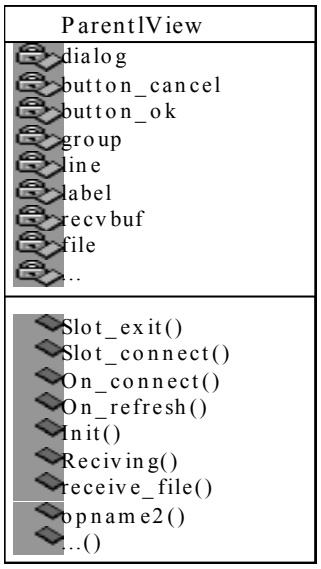
La classe VisualisatorDoc apporte les fonctionnalités de base des documents définis par l'utilisateur. Un document représente l'unité de données que l'utilisateur peut ouvrir et enregistrer respectivement avec les commandes file open et file save.

Cette classe supporte les opérations standards comme le chargement, la création et l'enregistrement de document.

L'utilisateur interagit avec la classe VisualisatorDoc avec l'intermédiaire de la classe ParentView. En effet cette dernière visualise une image du document et interprète les entrées de l'utilisateur comme des opérations sur le document.

Fig. 19 : La classe VisualisatorDoc

6.3.2.3 La classe ParentView



La classe ParentView représente la vue principale de l'application à laquelle peuvent être attachées plusieurs autres vues qui sont les ChildView. Cette classe instancie les classes ChildView et les attachent par la fonction add_child().

La classe ParentView instancie aussi la classe RecivingThread qui permet de recevoir des informations de l'agent global après connexion.

Les méthodes de cette classe permettent de mettre en œuvre les actions de l'utilisateur reçues de la classe VisualisatorApp.

Cette classe permet aussi de mettre à jour les Documents de l'application à chaque fois qu'il y a modification, en communiquant les informations à la classe VisualisatorDoc.

Fig. 20 : La classe ParentView

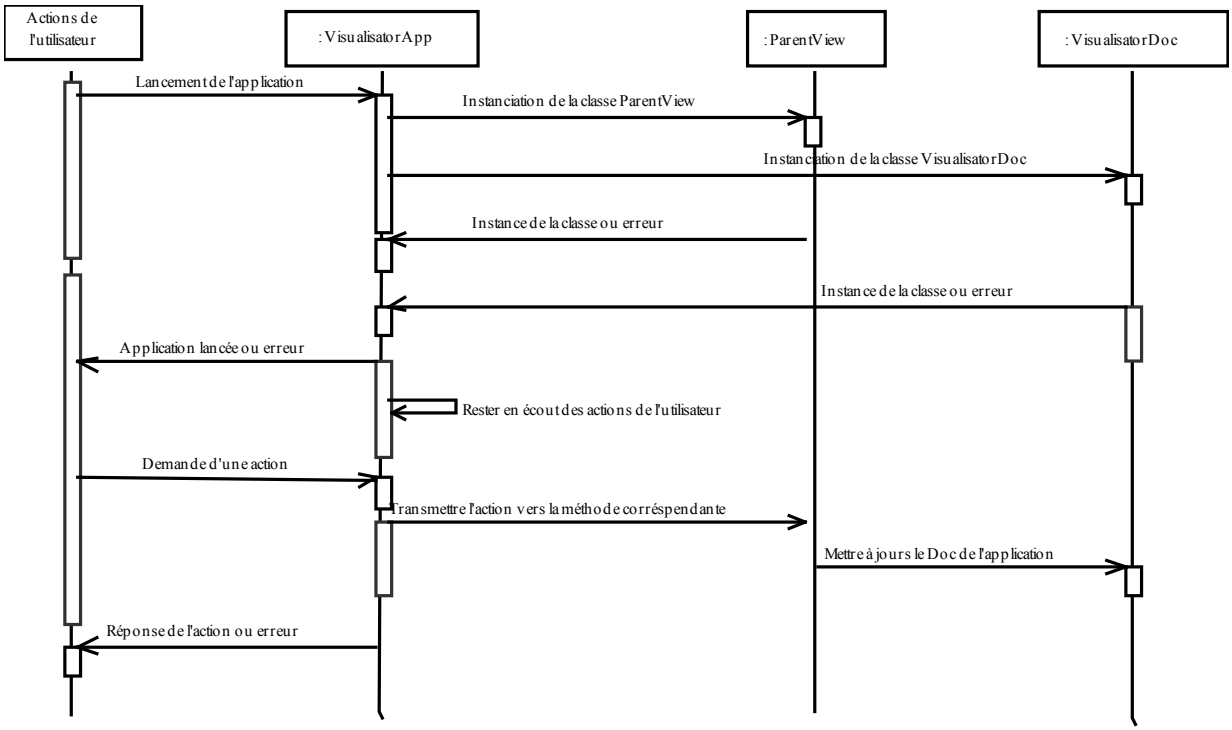
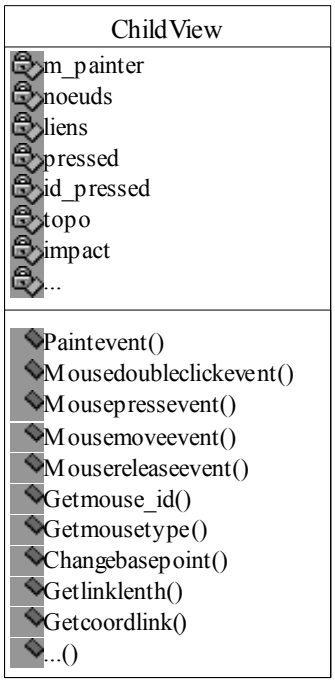


Fig. 21 : Diagramme des interactions entre classes

6.3.2.4 La classe ChildView



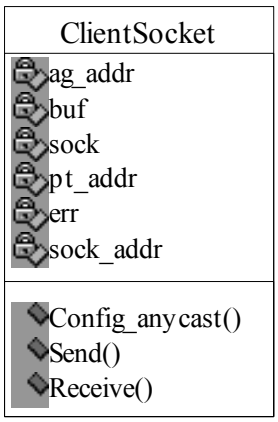
Cette classe représente la vue d'œuvre sur laquelle va agir l'utilisateur. De ce fait, elle est considérée comme intermédiaire entre l'application et ses utilisateurs. Elle permet d'afficher la topologie et peut servir comme moyen d'acquisition ou de demande d'informations à travers des boites de dialogues.

Pour atteindre ses objectifs, la classe ChildView fait appel à d'autres classes à savoir ClientSocket, Parser et Placement. Elle les instancie et utilise leurs méthodes pour les différentes opérations de calcul, d'affichage et de communication avec l'agent global

Les différentes méthodes de cette classe permettent la gestion des périphériques comme la souris et le clavier à fin de répondre aux besoins de l'utilisateur et mettre en œuvre ses actions.

Fig. 22 : La classe ChildView

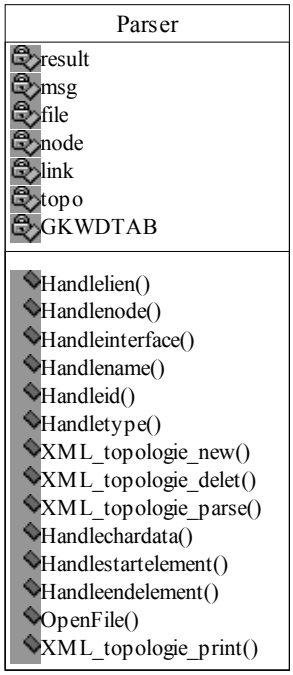
6.3.2.5 La classe ClientSocket



Cette classe représente le moyen avec lequel l'agent de visualisation communique avec l'agent global. En effet les méthodes de cette classe permettent de créer une socket v6 et d'initialiser ses paramètres à savoir la famille d'adresse, le protocole utilisé, l'adresse elle-même et le numéro de port puis d'effectuer les opérations d'envoi et de réception de données en utilisant les primitives Sendto() et Recvfrom().

Fig. 23 : La classe ClientSocket

6.3.2.6 La classe Parser



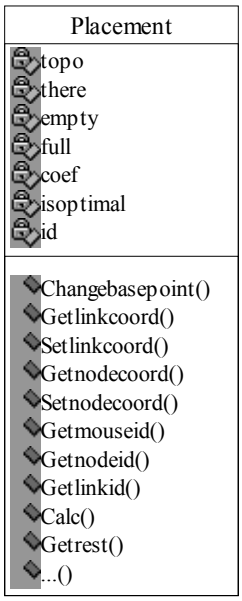
Cette classe représente un parseur de fichier XML. Grâce à ses méthodes, elle permet de parcourir un fichier et dégager les différentes balises qui s'y trouvent et stocke le résultat dans une structure de données correspondante.

Selon la spécification du fichier, déjà précisée auparavant, les méthodes de cette classe permettent de dégager les nœuds, les liens, les interfaces et leurs informations. Ces opérations sont assurées par les fonctions Handlelien, Handlenode, Handleinterface, Handleid …

Le résultat du parsing du fichier est stocké dans la structure de données result, qui va contenir toutes les informations de la topologie du réseau. Ces informations seront fournies après à la classe Placement qui va calculer les coordonnées des éléments selon un algorithme bien précis afin d'assurer un affichage claire de la topologie.

Fig. 24 : La classe Parser

6.3.2.7 La classe Placement



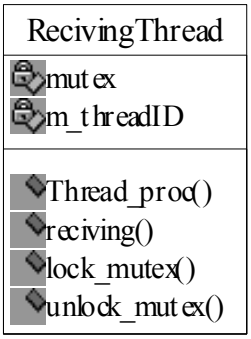
Les méthodes de cette classe permettent d'effectuer les différentes opérations de calcul des positions des nœuds et des liens.

En effet une fois les informations de la topologie stockées dans les structures de données correspondantes, cette classe va les prendre comme entrée et va déterminer les coordonnées des différents objets suivant un algorithme implémenté à travers ses méthodes.

Pour mémoriser les positions des différents éléments du réseau nous avons utilisé des structures de données sous forme de liste. Nous avons donc une liste de liens et une liste de nœuds dans lesquelles se trouvent toutes leurs informations et leurs coordonnées. De cette manière à chaque manipulation de l'utilisateur, la liste correspondante sera parcourue afin de mettre à jour les coordonnées de l'élément qui vient d'être déplacé.

Fig. 25 : La classe Placement

6.3.2.8 La classe RecivingThread



Cette classe permet la création d'un thread qui a pour rôle de rester en écoute sur le réseau, afin de recevoir des informations de l'agent global. Ce processus doit tourner en arrière plan et doit être transparent pour l'utilisateur.

Pour ne pas tomber dans une situation d'exclusion mutuelle, qui vient du fait que le processus d'écoute et celui de la manipulation de l'application par l'utilisateur accèdent aux mêmes données et les mêmes ressources, nous avons utilisés des mutex.

Fig. 26 : La classe RecivingThread

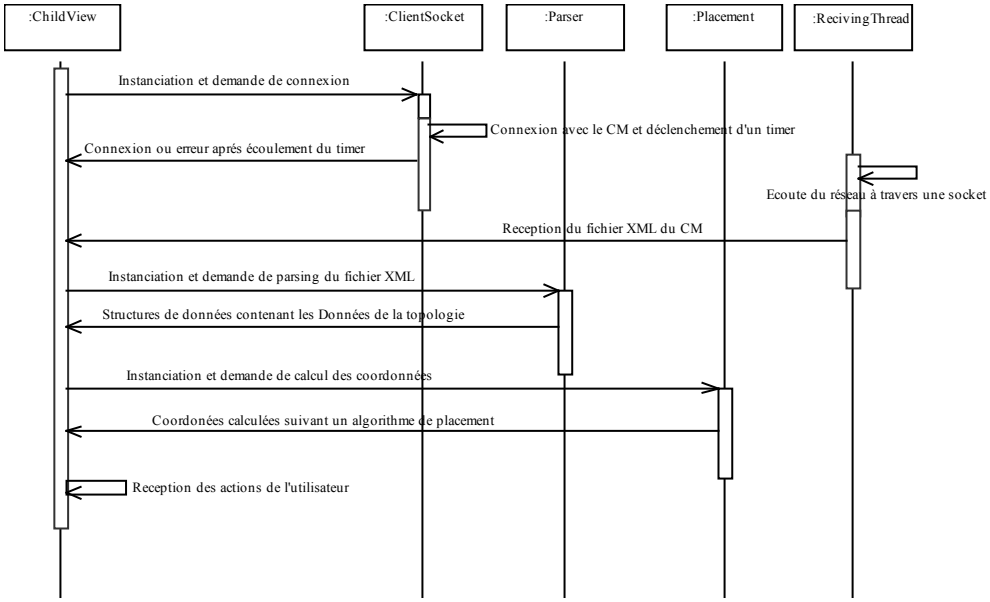


Fig. 27 : Diagramme d'interaction entre les classes

6.2 Conclusion

La conception de l'application a du passer par deux étapes principales, celle de la partie agent global qui se traduit par des modules intégrés dans l'ancien travail pour assurer la communication avec les agents de visualisation et celle des agents de visualisation eux même dans laquelle nous avons procédé par une conception orientée objet pour une bonne modélisation du problème et pour répondre aux besoins de l'application.

Chapitre VII

LA REALISATION

7.1 Introduction

Pour mettre au point un outil de visualisation de topologie des réseaux IPv6, nous avions à notre disposition une plateforme FreeBSD 4.5, composée d'un nombre de nœuds et de routeurs supportant le protocole IPv6.
Les machines utilisées sont des DELL Pentium 2 et 3
La structure de la salle IPv6 est la suivante :

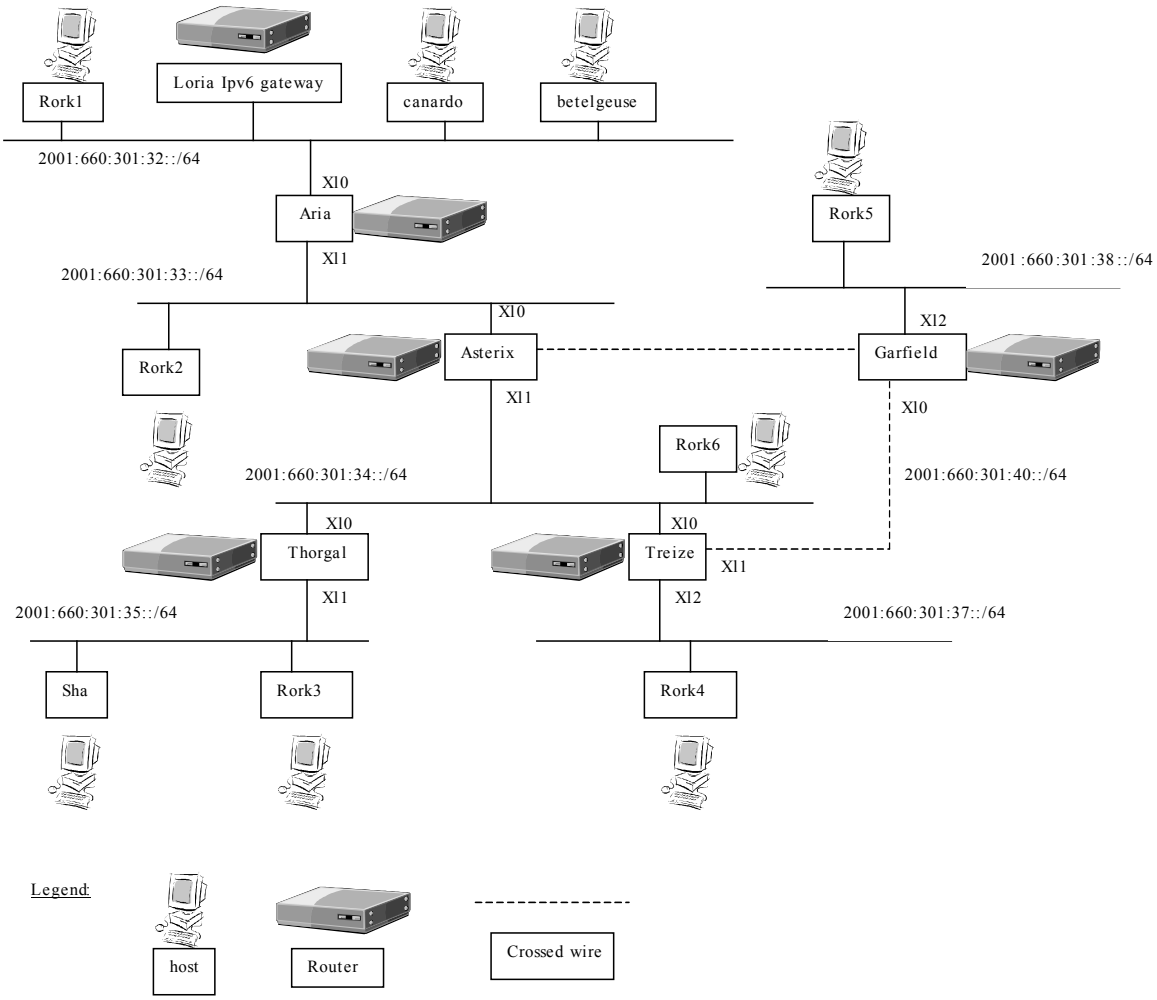


Fig. 28 : Architecture de la plateforme de travail

7.2 Choix des outils

Vu que l'environnement de développement de notre application est Unix FreeBSD, il a fallut chercher un outil graphique compatible avec ce genre d'environnement. Nous avons donc choisit au début la bibliothèque GDK avec le langage C.

En procédant à la conception de notre application nous nous sommes rendus compte que le modèle objet était beaucoup plus adéquat pour modéliser nos besoins et présente des apports très bénéfiques surtout de point de vue gestion des interactions entre objets et leur manipulation, pour cette raison nous sommes passés dans une seconde étape à l'utilisation de la bibliothèque GDK avec le langage C++.

Mais à travers des constatations récoltées d'après des tests et des participations à des mailing lists spécialisées nous avons constatés que cette bibliothèque était trop limitée et ne permet pas d'assurer les fonctionnalités espérées à travers l'application d'où l'adoption dans une dernière étape de la bibliothèque QT avec le langage C++. Cette dernière bibliothèque permet non seulement de nous fournir un environnement de développement adéquat pour notre application mais aussi la particularité d'être portable sur différents environnements : Windows 95/98/NT/2000, Linux, Solaris, HP-UX et plusieurs versions de Unix avec X11. Par ailleurs la version 2.3.1 de QT a été utilisée.

Le langage utilisé est le langage C++ car il est reconnu pour sa productivité, la facilité de sa maintenance et sa souplesse. Ce langage est idéal car il est proche du problème à résoudre de manière que les concepts d'une solution puissent être exprimés directement de façon concise.

7.3 Mise en œuvre

Pour mettre en œuvre notre application il a fallut dans une première étape comprendre l'esprit de l'application existante et l'analyser afin d'y intégrer notre travail. Cette tache présentait quelques difficultés à cause de la complexité des algorithmes et le très grand nombre de lignes de code qui se comptent en millier.

Ensuite nous sommes passé a la préparation de la plateforme de travail en installant les différents outils que nous devions utiliser, tous ces outils sont des logiciels libres disponible sur le site www.FreeBSD.org.

Une fois la plateforme prête, nous avons procédé à l'implémentation de notre application qui se divise en deux parties : celle qui doit être intégrée dans l'agent global et celle de l'agent de visualisation proprement dit.

7.3.1 Contribution au niveau de l'agent global

La contribution au niveau de l'agent global consiste à y intégrer des modules qui doivent permettre d'assurer des communications avec les agents de visualisation et d'autres modules permettant de collecter les données de la topologie du réseau reçues des agents de visualisation et les formater pour les envoyer.

Nous allons expliquer dans ce qui suit comment s'effectue la collection des informations de la topologie lorsqu'on reçoit plusieurs vecteurs de données en provenance de plusieurs agents locaux.

La fonction qui permet d'assurer cette tache est la fonction remp_struct qui est une fonction récursive qui prend comme entrée, au début, le nœud courant c'est-à-dire l'agent global et elle étudie chacune de ses interfaces. Pour chaque interface la fonction cherche la BDLA qui contient cette interface. La BDLA (base de données de l'agent local) contient l'ensemble des interfaces qui se trouvent sur le même lien que l'agent local. Ainsi nous découvrons un lien avec les interfaces qui lui sont reliées. En plus l'agent global effectue une requête vers le serveur DNS (**D**omain **N**ame **S**erver) afin de connaître le nom de la machine à laquelle appartient l'interface. En effet le serveur DNS, consulte sa table de correspondance et déduit le nom de la machine à partir de l'adresse de l'interface. De cette manière nous pouvons associer les interfaces d'un même nœud.

Lorsque la fonction remp_struct() est appelée, elle étudie chacune des interfaces du nœud local. Pour chacune d'elles, elle cherche les interfaces qui se trouvent sur le même lien qu'elle. Si une interface d'entre elles appartient à un nœud qui possède plusieurs interfaces, la fonction remp_struct est relancée en prenant comme entrée ce nouveau nœud, d'où le lancement récursif de l'algorithme. Ainsi toutes les interfaces et les liens seront parcourus et découverts.

L'algorithme de cette fonction est le suivant :

Traitement du nœud local (sur lequel se trouve l'agent global)

Chercher ses interfaces une par une

Pour chaque interface chercher l'agent local qui l'a découvert

Consulter la BDLA de l'agent local et dégager toutes les interfaces qu'il a pu découvrir.

Pour chaque interface si elle appartient à un nœud comportant plus qu'une interface, relancer l'algorithme en donnant ce nœud comme entrée.

Sinon il y a appel à la fonction add_node() qui permet de traiter l'interface trouvée et enregistrer ses informations : adresse physique, globale, lien local, nom …

Traitement de la prochaine interface de la BDLA

Traitement de la prochaine interface du nœud fournit en entrée

Fin de l'algorithme.

Il reste à signaler que cet algorithme fait appel à d'autres fonctions permettant de faire l'identification des liens grâce aux interfaces qui lui sont reliées, l'attribution

d'identificateurs uniques pour les liens, les interfaces et les nœuds et la vérification de la cohérence des données.

Une fois la collecte des données réalisée, la fonction construir_XML() est lancée afin de formater les données en XML.

La figure Fig. 28 représente un exemple de fichier XML généré par l'agent global selon la spécification que nous avons précisée dans le chapitre précédent.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE IPv6:topologie>
<IPv6:topologie xmlns:IPv6="">

<lien>
        <id>1</id>
</lien>
<node>
        <name>Garefield </name>
        <id>1</id>
        <type>router </type>
        <link>1</link>
        <interface>
                <name>xl0 </name>
                <id>11</id>
                <phys_addr>0-1-2-e3-60-89-/6 </phys_addr>
                <global_addr>2001:660:301:32:201:2ff:fee3:6089 </global_addr>
                <link_addr>unknown </link_addr>
        </interface>
        <interface>
                <name>xl1 </name>
                <id>12</id>
                <phys_addr>0-1-2-e3-60-2f-/6 </phys_addr>
                <global_addr>unknown </global_addr>
                <link_addr>unknown </link_addr>
        </interface>
        <interface>
                <name>xl2 </name>
                <id>13</id>
                <phys_addr>0-1-2-e3-60-5-/6 </phys_addr>
                <global_addr>unknown </global_addr>
                <link_addr>unknown </link_addr>
        </interface>
</node>
</IPv6:topologie>
```

Fig. 29 Exemple de fichier XML

7.3.2 Réalisation de l'agent de visualisation

Pour la réalisation de l'agent de visualisation nous avons procédé par une conception orientée objet à travers laquelle nous avons essayé de ramener les objets du monde réel (machines, nœuds, sous réseaux …) en objets abstraits dans notre langage de programmation. En effet chaque élément de la topologie du réseau est modélisé par une instance d'objet de type CHost, CLink ou CRouter. Ces objets seront dégagés à partir du fichier XML envoyé par l'agent global vers l'agent de visualisation. La lecture du fichier XML se fait par un analyseur implémenté par la classe Parser.

Dans ce qui suit nous allons expliquer le principe de fonctionnement de l'agent de visualisation en faisant la correspondance entre les objets détaillés dans la conception et les éléments de l'application réelle.

7.3.2.1 Connexion à l'agent global

Pour la connexion à l'agent global nous avons choisit l'utilisation des adresses anycast, l'utilisateur n'a qu'a fournir cette adresse pour demander une connexion. L'adresse utilisée dans notre application est : 2001:660:301:32::a.

La gestion des connexions et des communications se font à l'aide des méthodes de la classe ClientSocket. La configuration de l'interface socket se fait comme suit :

```
int sock ;
struct sockaddr_in6 sd;
sock=socket(AF_INET6, SOCK_DGRAM, IPPROTO_UDP);
sd.sin6_len=sizeof(struct sockaddr_in6);
sd.sin6_family=AF_INET6;
sd.sin6_port = htons(57410);
inet_pton(AF_INET6,«2001:660:301:32::a»,(struct in6_addr*)&sd.sin6_addr.s6_addr[0] )
```

L'utilisateur fournit l'adresse anycast à travers une boite de dialogue (cf fig. 29) :

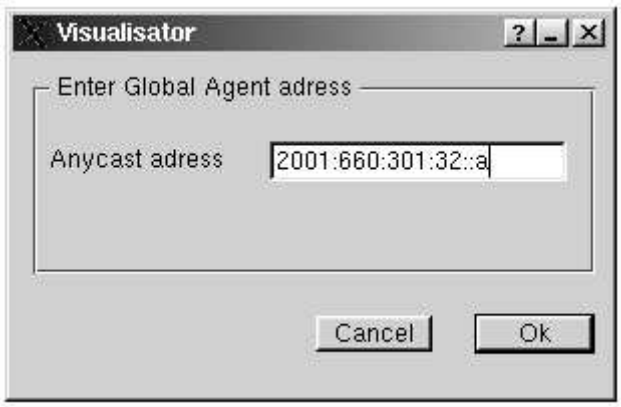


Fig. 30 : connexion à l'agent global

7.3.2.2 Méthode de calcul des coordonnées

Le placement des différents objets sur l'écran se fait à l'aide d'un algorithme de placement. La zone d'affichage de l'écran est gérée par une instance de la classe ChildView qui représente une des vues de l'application.

Le principe de l'algorithme d'affichage est la suivante :

L'espace réservé à la visualisation de la topologie est modélisé par une matrice $M_{i,j}$ dont chaque élément M[i][j] constitue un endroit dans lequel on peut placer un élément du réseau de coordonnées i et j dans le repère de l'écran.

Le principe consiste à remplir les éléments de la matrice en évitant les superpositions des éléments et en tenant compte des groupements des éléments qui se traduit par exemple par l'appartenance de plusieurs nœuds à un même sous réseau.

Cet algorithme est implémenté à l'aide des méthodes de la classe Placement, qui permettent de faire tous les calculs de coordonnées et les optimisations nécessaires. Cela n'empêche pas que l'utilisateur peut déplacer tous ces objets par de simples cliques de la souris, pour cela nous avons du gérer les différents évènements de la souris à savoir MouseMoveEvent, MouseReleaseEvent, MousePressEvent, MouseClickEvent …

La matrice M est mise à jour à chaque modification de la disposition des objets où lors de l'apparition de nouveaux objets suite à une nouvelle découverte de la topologie.

7.3.2.3 Réception des données

En même temps que l'utilisateur manipule la topologie, un autre processus fonctionne en arrière plan afin de recevoir les mises à jour, qui proviennent de l'agent global, et qui traduisent l'évolution de la topologie au cours du temps. Le processus en question est un thread ou processus léger implémenté à l'aide de la classe RecivingThread. Cette classe utilise les fonctions de la librairie pthread qui se trouve dans pthread.h qui permettent la création, l'initialisation et la gestion des routines des threads en général.

Suite à la réception de nouvelles informations concernant la topologie, cette dernière sera réaffichée afin que l'utilisateur puisse suivre en temps réel l'évolution de la topologie du réseau.

Comme plusieurs processus tournent en même temps (le processus d'écoute et celui de l'interface avec l'utilisateur), nous étions confronté à un problème classique résultant de l'utilisation des threads, qui est le problème de concurrence d'accès aux données où le problème d'exclusion mutuelle. En effet les deux processus accèdent aux même structures de données que se soit en lecture/écriture ou des lectures simultanée, ce qui nous a amené à utiliser des mutex pour le blocage «l ock » et libération « unlock » des ressources.

7.3.3 Tests de l'application

Lors du lancement de l'application, une interface graphique apparaît, à travers laquelle l'utilisateur a la possibilité de se connecter à l'agent global grâce à une boite de dialogue qui apparaît en appuyant sur le bouton connect. L'utilisateur n'a donc qu'à fournir l'adresse anycast de l'agent global. Ainsi l'agent de visualisation recevra les informations concernant la topologie et les affiche.

La figure représente un exemple d'exécution de l'application visant à découvrir la topologie de la plateforme IPv6 du LORIA.

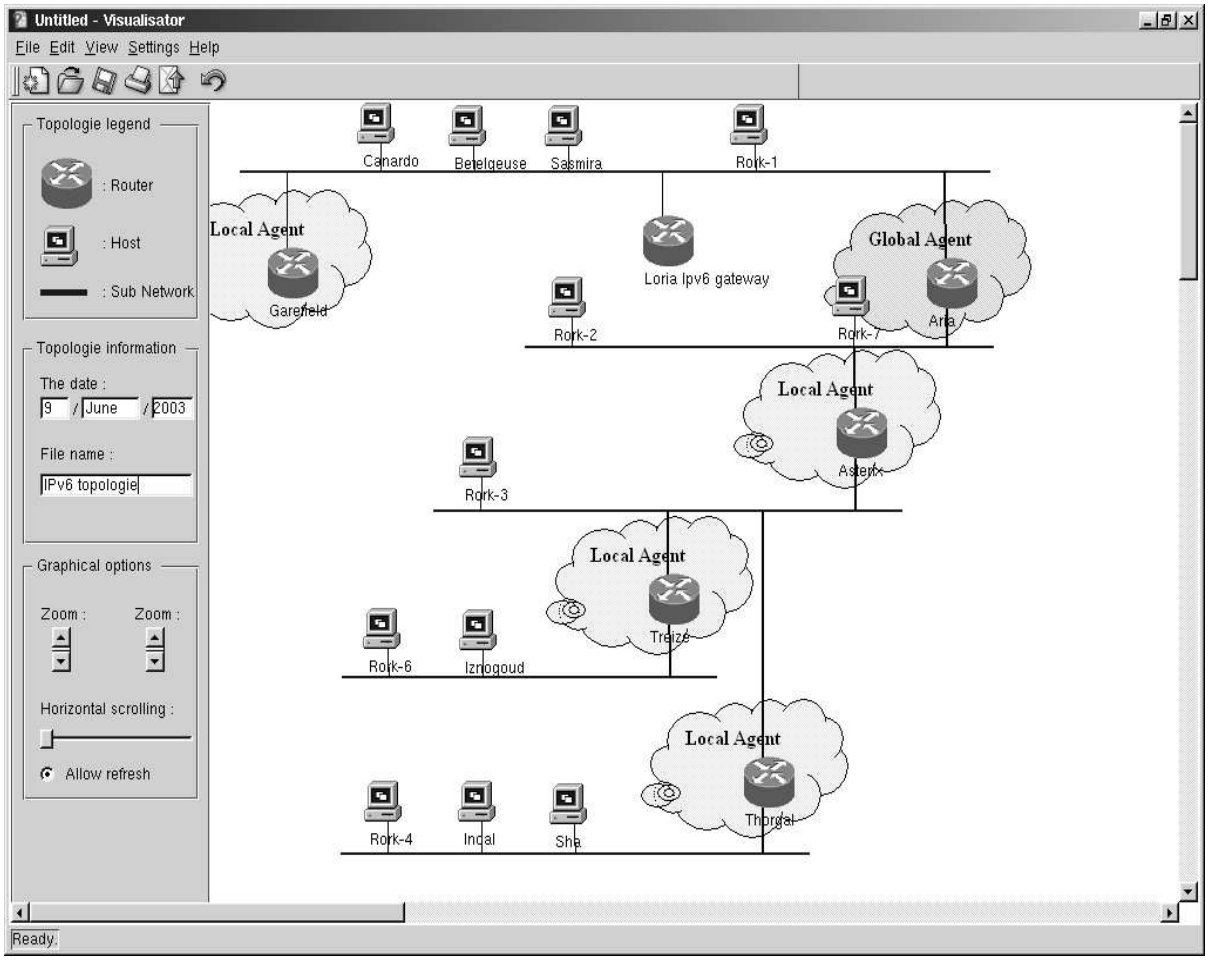
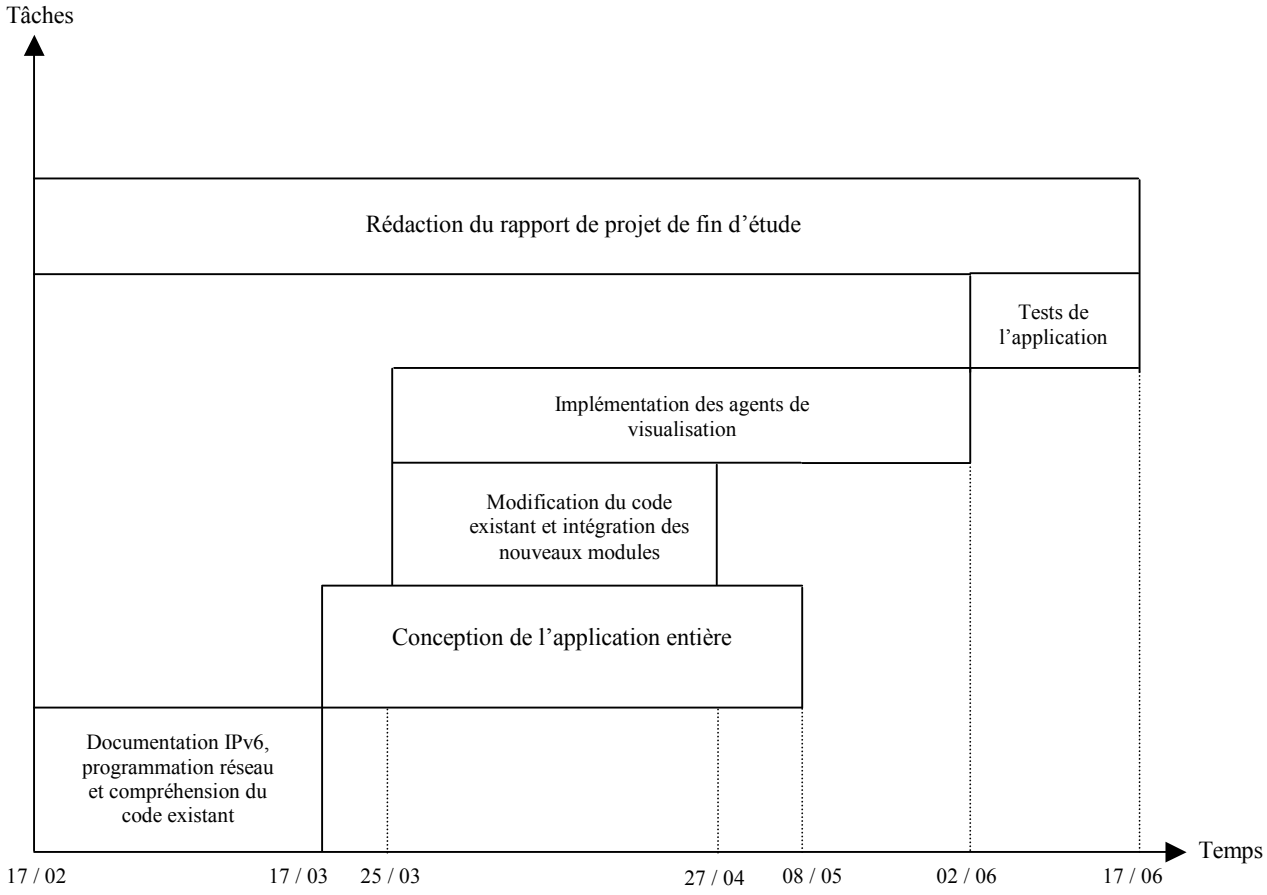


Fig. 31 : Découverte de la topologie de la plateforme IPv6 du LORIA

7.4 Chronogramme du stage



7.5 Conclusion

Au terme de ce travail nous avons abouti à la réalisation d'un outil de visualisation de topologies pour les réseaux IPv6. Cet outil permet à un ou plusieurs utilisateurs de découvrir la topologie d'un réseau utilisant le protocole IPv6 et de suivre son évolution en temps réel au cours du temps. L'application a été testée sur la plateforme IPv6 du LORIA.

Chapitre VIII

CONCLUSION ET PERSPECTIVES

8.1 Conclusion

Le nouveau protocole IPv6, comme toute nouveauté, a besoin de services qui lui sont propres ou nécessite de faire évoluer d'anciens services pour qu'ils puissent continuer à être utilisés. L'un des services les plus importants, surtout pour la supervision et la gestion des réseaux, est le service de découverte de topologies qui permet d'avoir une vue globale sur le réseau et permet de suivre dynamiquement son évolution au cours du temps. Bien que pour le protocole IPv4 plusieurs outils existent, les choses dans le monde IPv6 ne sont pas aussi abouties du fait qu'il n'existe qu'un seul outil et qui n'est pas fonctionnel en milieu hétérogène.

Durant ce stage nous avons essayé de compléter un travail qui a déjà débuté visant à réaliser un outil propre au protocole IPv6 est qui est fonctionnel sur toute plateforme. Le travail consiste en la conception et la réalisation d'une infrastructure de visualisation des réseaux utilisant le nouveau protocole. Notre application doit permettre la visualisation de la topologies des réseaux à l'aide des informations récoltées par un algorithme de recherche distribué qui fait la découverte de topologie. Pour ce faire nous avons du suivre l'esprit de sa conception qui repose sur un modèle hiérarchique constitué de deux niveaux : les agents locaux qui récoltent les informations sur les sous réseaux et l'agent global qui gère ces agents locaux. Nous avons donc ajouté un troisième niveau qui sont les agents de visualisations qui vont compléter la chaîne de travail afin d'aboutir au résultat attendu qui est l'affichage de la topologie du réseau pour l'utilisateur.

Notre outil a été testé sur une plateforme comportant des machines qui utilisent le protocole IPv6, ainsi il a été validé. En effet nous avons eu des résultats conformes à l'architecture de la plateforme avec une évolution dynamique au cours du temps.

8.2 Connaissances acquises

Ce stage m'a permis de découvrir le monde de la recherche, de m'intégrer dans l'équipe MADYNES et de me familiariser avec les travaux qu'elle mène actuellement. D'un autre côté, j'ai pu toucher de près à la programmation avec le langage C et C++ sous le système d'exploitation FreeBSD, ce qui m'a permit de consolider mes connaissances sur ces langages.

Je tiens aussi à dire, que c'était pour moi une grande occasion de travailler sur un vrai système informatique en réseau, où tout est accessible de partout avec une transparence et une fiabilité garantie, ce qui a permis de mettre en application les connaissances que j'ai pu acquérir tout au long de ma formation.

8.3 Perspectives

A la fin de notre travail nous avons abouti à un outil qui permet la découverte de topologie des réseaux. Pour l'amélioration de l'application nous pouvons envisager comme perspective l'utilisation des réseaux actifs pour le lancement des agents locaux, chose qui se fait actuellement manuellement.

Nous pouvons envisager aussi d'ajouter à cet outil des fonctions de gestion à condition d'utiliser le protocole SNMP et la MIB II, chose qui n'est pas garantie maintenant à cause de la non disponibilité de ces services avec le protocole IPv6. De cette manière on peut aboutir à un outil très intéressant pour la gestion des réseaux conforme aux outils existants pour le protocole IPv4 qui sont largement utilisés par les administrateurs de réseaux.

BIBLIOGRAPHIE

[CD98] A. Conta and S. Deering. Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification. Technical report, IETF, December 1998. RFC 2463.

[CFM99] R. Coltun, D. Ferguson, and J. Moy. OSPF for IPv6. Technical report, IETF, December 1999. RFC 2740.

[CFSD90] J.D. Case, M. Fedor, M.L. Schoffstall, and C. Davin. Simple network management protocol (snmp). Technical report, IETF, May 1990. RFC 1157.

[Cra02] M. Crawford. "ipv6 node information queries, draft-ietf-ipngwg-name-lookups 09.txt". Technical report, IETF, 2002.

[DFh99] S. Deering, W. Fenner, and B. haberman. Multicast Listener Discovery (MLD) for IPv6. Technical report, IETF, October 1999. RFC 2710.

[DLR+93] E. Decker, P. Langille, A. Rijsinghami, , and K. McCloghrie. Definition of Managed Objects for Bridges. Technical report, IETF, July 1993. RFC 1493.

[HPView] http://www.openview.hp.om.

[McC96a] K. McCloghrie. Management Information Base for the Internet Protocol (IP).Technical report, IETF, November 1996. RFC 2011.

[McC96b] K. McCloghrie. Management Information Base for the Transmission Control Protocol (TCP). Technical report, IETF, November 1996. RFC 2012.

[McC96c] K. McCloghrie. Management Information Base for the User Datagram Protocol (UDP). Technical report, IETF, November 1996. RFC 2013.

[MM97] G. Malkin and R. Minnear. RIPng for IPv6. Technical report, IETF, January 1997. RFC 2080.

[MP85] J.C. Mogul and J. Postel. Internet Standard Subnetting Procedure, RFC 950,STD 5, August 1985.

[MPS99a] K. McCloghrie, D. Perkins, and J. Schoenwaelder. Structure of ManagementInformation v2 (SMIv2). Technical report, IETF, April 1999. RFC 2578, STD 58.

[MPS99b] K. McCloghrie, D. Perkins, and J. Schoenwaelder. Textual conventions for SMIv2. Technical report, IETF, April 1999. RFC 2579, STD 58.

[NNS98] T. Narten, E. Nordmark, and W. Simpson. Neighbor Discovery for IP Version 6 (IPv6). Technical report, IETF, December 1998. RFC 2461.

[Pos81] J. Postel. Internet Control Message Protocol, RFC 792, STD 5, September 1981.

[Ros 94] Rose (Marshall T.) -The Simple Book : An Introduction to Management of TCP/IP-based Internets. Second Edition.- Prentice Hall.

[Son 99] P. Sonntag - Gestion IPv6 : Mémoire en vue d'obtention du diplôme ingénieur CNAM. Février 1999.

[Ste 90] D. Steedman : Abstract Syntax Notation one (ASN.1) : The tutorial and reference. Technology Appraisals Ltd, 1990.

[Ste 96] W. Richard Stevens : TCP/IP illustré : Les protocoles. Volume1, traduction de Eric Tyberghien. International Thomson Publishing, 1996.

[Ste 98] W. Richard Stevens : UNIX Network Programming, Networking APIs : Sockets and XTI. Volume1, Second édition 1998.

[SZ 97] Siomoni (Noemie) et Znaty (Simon). - Gestion de réseau et service. - InterEditions, 1997. ISBN 2-225-82980-2.

ANNEXE A

GUIDE D'UTILISATION

Pour la visualisation de la topologie d'un réseau IPv6 à l'aide de notre outil de découverte, un utilisateur doit passer par trois étapes fondamentales :

1. Lancement des agents locaux

L'utilisateur doit lancer les agents locaux, cette opération se fait manuellement à travers la ligne de commande (cf Fig. 1).

La commande de lancement est ./Agentloc [nom de l'interface]. En effet chaque agent local est lancé sur une des interfaces du nœud sur lequel il est exécuté. Ainsi l'agent local va découvrir les nœuds se trouvant sur le même sous réseau que l'interface avec laquelle il est lancé. De ce fait le choix des agents locaux est très important puisque l'utilisateur doit lancer autant d'agents de visualisation que de sous réseaux.

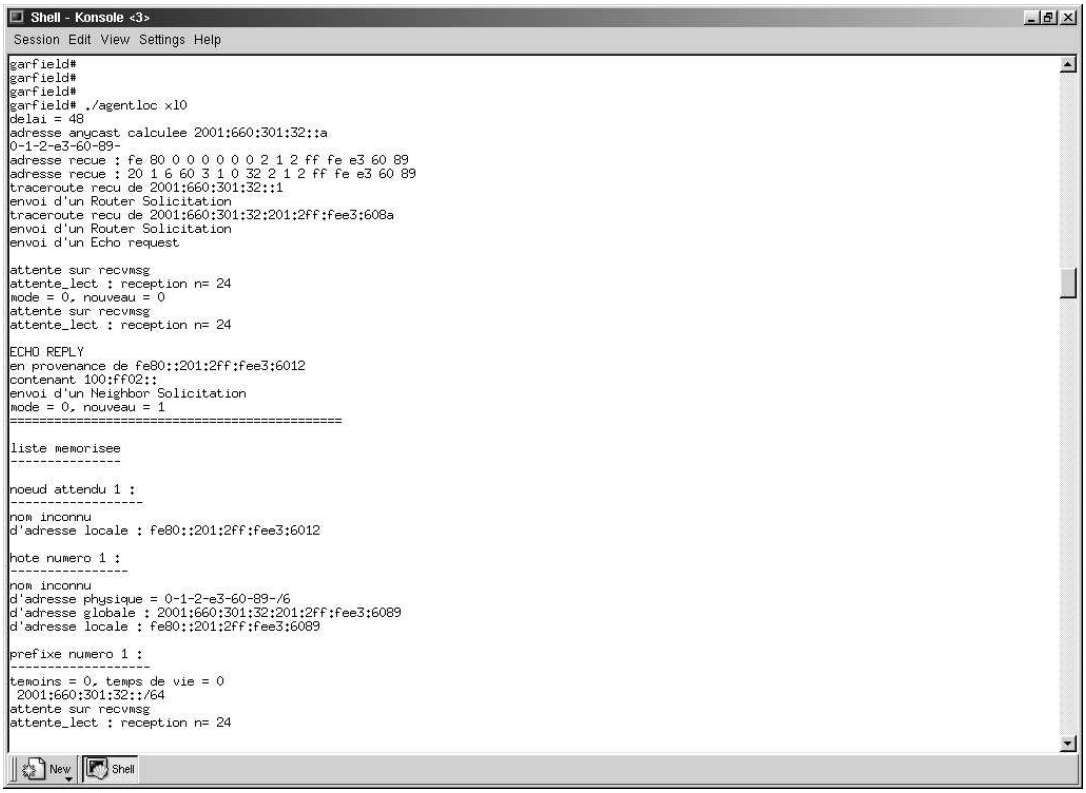


Fig.1 Lancement de l'agent local sur Garfield avec l'interface xl0

2. Lancement de l'agent global

L'utilisateur doit lancer l'agent global qui permet de récolter les informations des agents locaux. Le lancement de l'agent global se fait aussi de manière manuelle à travers la ligne de commande (cf Fig 2) avec la commande ./AgentCM.

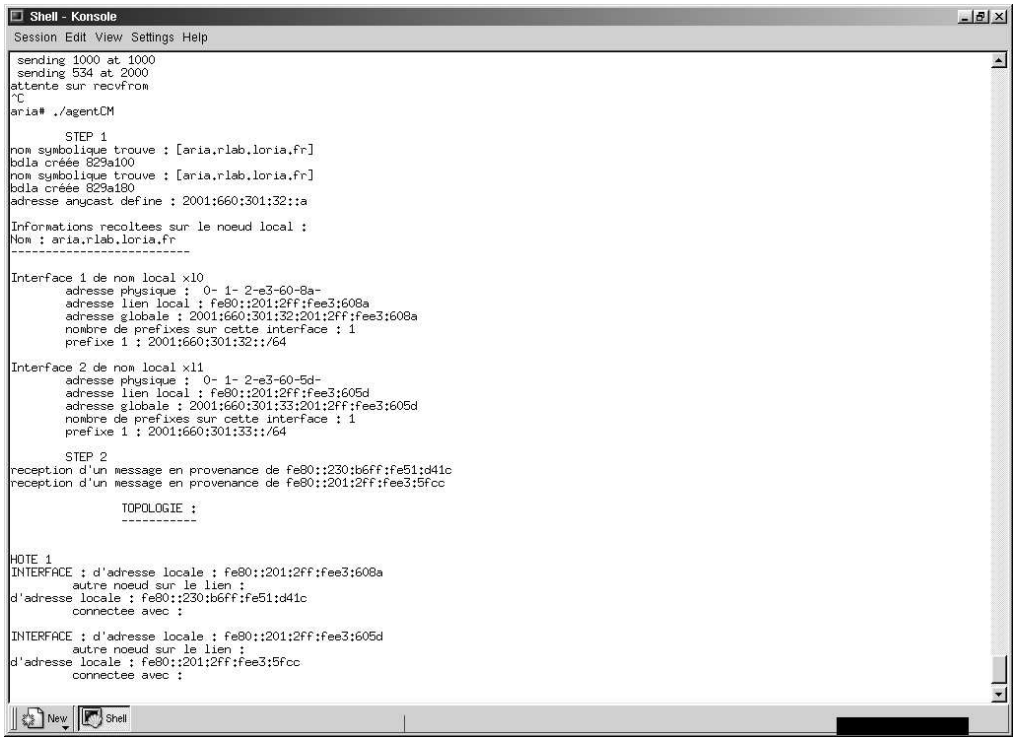


Fig. 2 Lancement de l'agent global

3. Lancement de l'agent de visualisation

Pour la visualisation de la topologie du réseau, l'utilisateur doit lancer l'agent de visualisation. Ainsi une interface graphique apparaît à travers laquelle l'utilisateur pourra se connecter en fournissant l'adresse anycast de l'agent global (cf Fig 3)

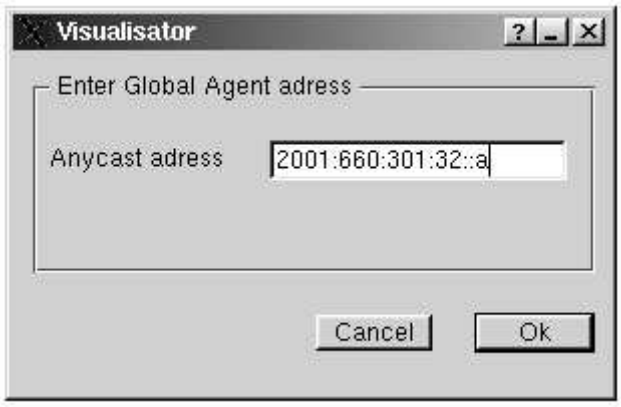


Fig.3 Connexion à l'agent global

Une fois que l'utilisateur aura fourni l'adresse, trois cas sont envisageable :

- ✓ L'adresse n'est pas une adresse reconnue, l'utilisateur sera aussitôt avertis par un message d'erreur,
- ✓ L'écoulement d'un timer sans aboutir à une connexion, l'utilisateur sera aussi avertis du refus de connexion,
- ✓ L'établissement de la connexion et affichage de la topologie (cf Fig 4),

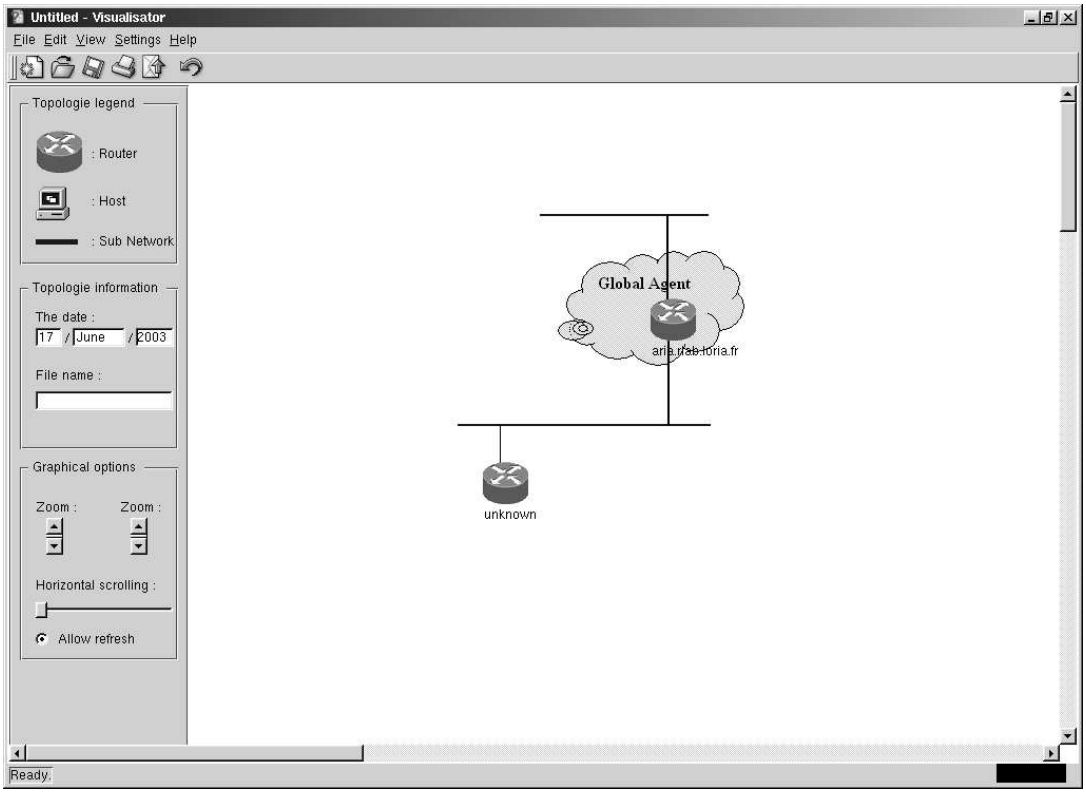


Fig. 4 Etablissement de la connexion

La figure 4 représente l'interface à travers laquelle l'utilisateur va afficher la topologie. Dans cette figure la topologie se résume à l'agent global, un autre nœud découvert et deux sous réseaux seulement car l'agent global n'a pas encore reçu d'informations des agents locaux, d'où il ne connaît que lui-même, les deux liens sur lesquels il se trouve et un voisin qu'il a pu découvrir.

L'agent global reste en écoute sur le réseau pour recevoir des informations en provenance des agents locaux ou d'autres demandes de connexions d'agents de visualisation. Dés la réception de nouvelles données concernant la topologie, elles seront aussitôt envoyées vers les agents de visualisation, qui eux aussi reste en écoute sur le réseau. Ainsi la topologie sera mise à jours au cours du temps

La figure 5 montre l'apparition de nouveaux éléments, ce qui correspond aux informations reçues de l'agent local Garfield.

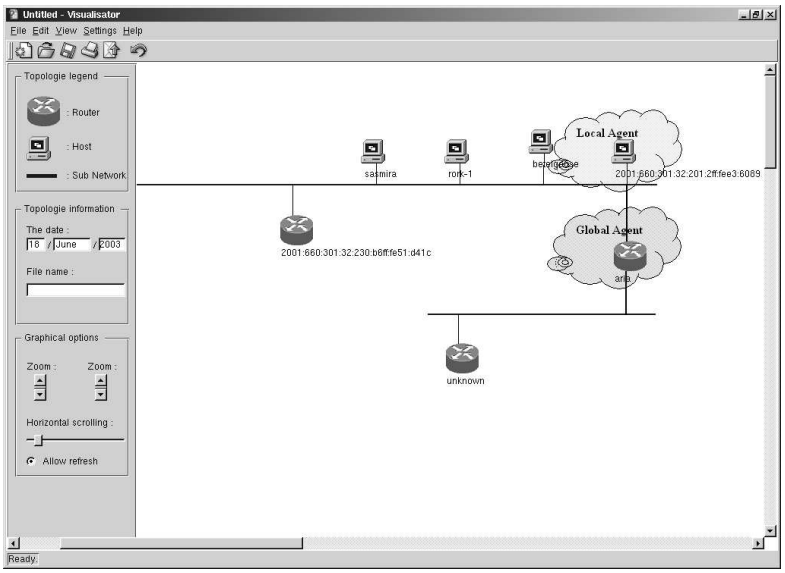


Fig. 5 réception de données en provenance de Garfield

La découverte de topologie se poursuit au cours du temps jusqu'a la découverte totale de la topologie (cf Fig. 6) et à chaque modification, l'agent de visualisation fera la mise à jours, ce qui permet à l'utilisateur de suivre en temps réel l'évolution de la topologie au cours du temps.

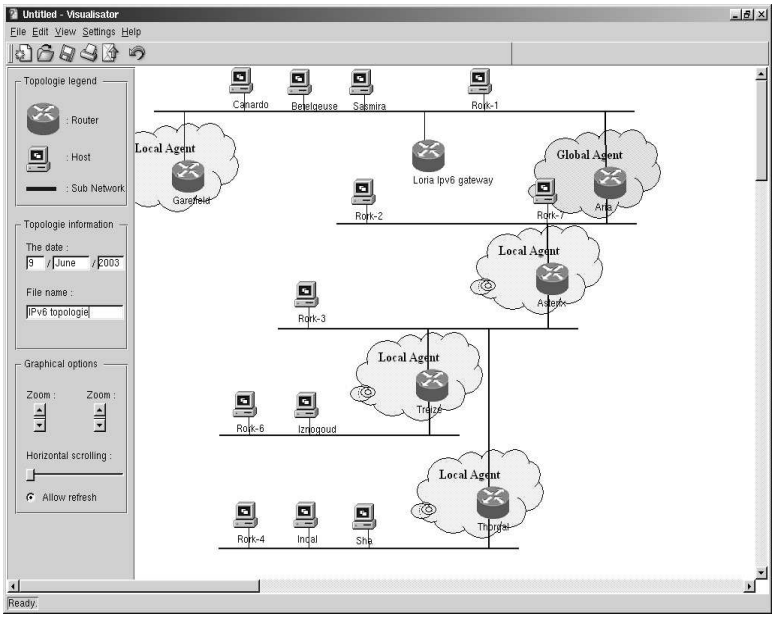


Fig. 6 Découverte totale de la topologie

A travers l'agent de visualisation, l'utilisateur peut effectuer plusieurs opérations à savoir la demande d'informations concernant des nœuds (cf Fig. 7).

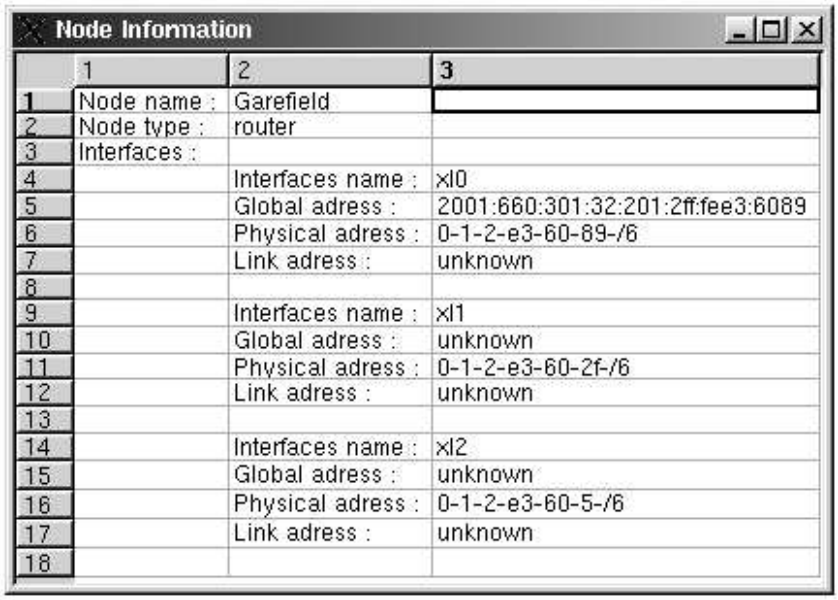


Fig. 7 Informations du nœuds Garfield

L'utilisateur peut aussi manipuler la topologie du réseau par de simple cliques de la souris ou effectuer des opérations de zoom : horizontal ou vertical. Il peut aussi permettre à l'agent de visualisation de recevoir des mises à jours ou non.

A chaque modification de la topologie, l'utilisateur a la possibilité de l'enregistrer. La demande d'enregistrement se fait automatiquement lors de la fermeture de l'application (cf Fig. 8).

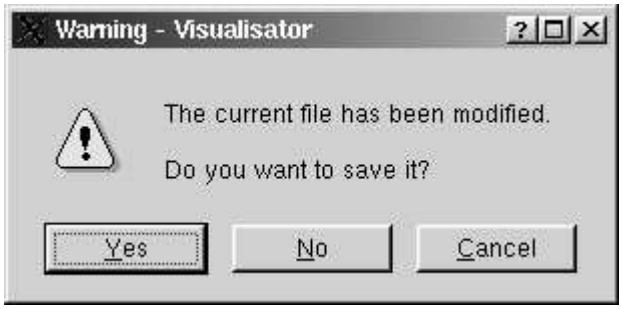


Fig. 8 Demande d'enregistrement

L'utilisateur peut aussi ouvrir des fichiers enregistrés contenant des topologies anciennes (cf Fig 9).

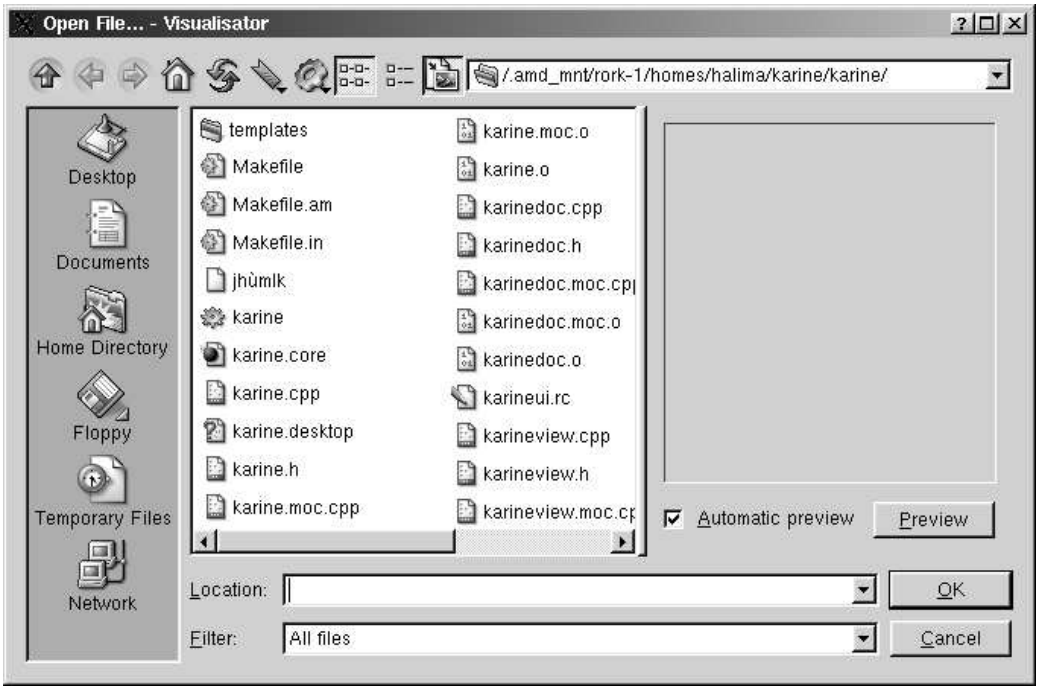


Fig. 9 Ouverture de fichiers

L'agent de visualisation permet aussi l'impression de la topologie (cf Fig. 10).
Toutes les fonctionnalités citées sont offerte grâce au menu et la barre d'outil qui contiennent des boutons permettant aussi d'assurer d'autres fonctionalités.

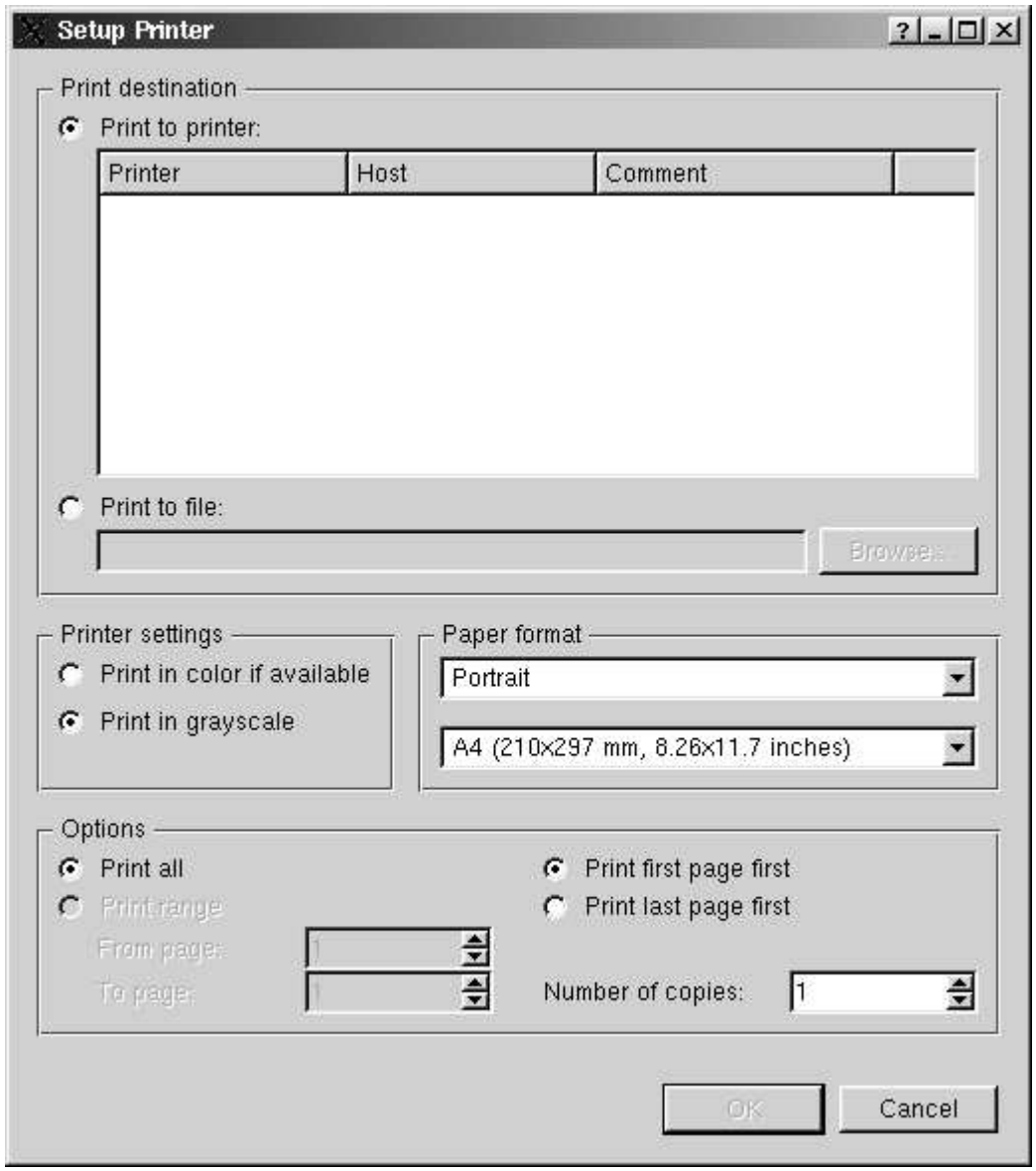


Fig. 10 Impression de la topologie

ANNEXE B

TRANSITION IPv4/IPv6

1. Introduction

La transition du protocole IPv4 vers IPv6 va se faire progressivement avec une longue période de cohabitation. Plusieurs solutions ont donc été développées pour :

- ✓ Assurer la communication entre des réseaux IPv4 et des réseaux IPv6
- ✓ Assurer la communication entre deux réseaux IPv6 éloignés en passant par le monde IPv4

Progressivement, la complexité de conserver un réseau IPv4 augmentera, alors que la mise en oeuvre d'un réseau IPv6 deviendra de plus en plus simple. La multiplication des bulles IPv6 dans un monde principalement IPv4 conduira vers un monde IPv6 où subsisteront des "bulles" IPv4.

2. La transition : Intégration ou migration ?

Le passage de IPv4 vers IPv6 se fera-t-il par une intégration (transition lente avec une cohabitation des deux systèmes) ou par une migration ? Une chose est sure, il n'y aura pas un jour J où l'ensemble du réseau devrait basculer. Pendant une durée qui pourra être assez longue, il y aura des "bulles" IPv6 dans un monde IPv4. Au fur et à mesure qu'IPv6 prendra de l'ampleur, la tendance s'inversera avec des bulles IPv4 restant dans un monde IPv6.

Il y aura une longue période de transition selon Bob Fink du Lawrence Berkeley Labs et co-président du groupe ngtrans en charge des recommandations pour la transition vers IPv6 à l'IETF (avec Alain Durand, un français actuellement chez Sun à Palo Alto).

3. Pourquoi l'effort pour la transition est significatif ?

Il y a des différences profondes entre les entêtes qui donnent les informations sur les paquets de données transportés dans IPv4 et IPv6 : les entêtes des paquets IPv6 sont de 40 octets. Les adresses de source et de destination du paquet prennent 4 fois plus de place que pour IPv4 pour une taille totale de l'entête qui n'est que deux fois plus grosse. Pour cela les autres champs de l'entête ont été largement modifiés pour être plus optimisés tout en permettant les nouveaux services (les champs optionnels d'IPv4 ne sont pas tous utilisés).

4. Les scénarios de cohabitation

Le groupe ngtrans propose plusieurs scénarios de transition et de cohabitation mais n'en impose aucun et laisse chacun choisir le plus adapté aux circonstances. Il y a deux grands types de problèmes à résoudre :

1. Passer d'un monde IPv6 à IPv4 et vice versa
2. Faire communiquer des réseaux IPv6 entre eux en passant par le monde IPv4

Lorsque la transition sera pratiquement complète, il faudra également prévoir de faire dialoguer des mondes IPv4 entre eux en passant par un monde IPv6.

5. Faire communiquer des serveurs IPv4 et IPv6

Il existe pour cela plusieurs solutions. Le plus souvent le système intègre les deux suites de protocoles IPv4 et IPv6 (on parle de piles de protocoles). Le RFC 1933 d'Avril 1996 définit la compatibilité de piles IPv4 et IPv6, mais il nécessite de configurer les deux piles, ce qui est complexe. De plus, cela ne solutionne pas le manque d'adresses IPv4 car chaque adresse IPv6 doit correspondre à une adresse IPv4. Il existe plusieurs façons plus récentes de faire cohabiter IPv4 avec IPv6.

Pour passer d'un monde IPv4 à IPv6 et vice-versa, il faut assurer une traduction des entêtes de paquets et des adresses. Il existe de nombreuses solutions :

- ✓ Par la traduction des entêtes de paquet : Il existe plusieurs protocoles possibles tels que SIIT (RFC 2765 Stateless IP ICMP Translator) ou NAT-PT (RFC 2766 Network Address Translator Protocol Translator). La traduction d'adresses (SIIT ou NAT-PT) permet de connecter des objets uniquement IPv6 dans un monde encore IPv4 (par exemple des téléphones portables ou des automobiles connectées)

- ✓ Par une communication au niveau des applications grâce à des ALGs (Application Layer Gateways) pour des traducteurs de DNS. Il s'agit également de gérer la conversion "nom de domaine adresse IP" de type A DNS en IPv4 en IPv6 (de type A6 ou AAAA)

- ✓ Par l'allocation temporaire d'une adresse IPv4 pour permettre à une machine IPv6 de converser avec une machine IPv4. On fabrique alors un tunnel IPv4 dans le réseau IPv6 (Le protocole DSTM, Dual Stack Transition Mechanism, permet de ne configurer la pile IPv4 que lorsque cela est nécessaire évitant le plus souvent la double configuration).

Pour permettre à une bulle IPv6 de communiquer avec le reste du monde IPv4, on installe donc souvent une double pile. Mais souvent cette double pile n'est installée que sur un serveur qui sert alors pour la conversion. Si on installe la double pile que sur les serveurs, ils seront les seuls à pouvoir dialoguer directement avec le monde extérieur. Par exemple, les postes de travail qui n'auront qu'une seule pile ne pourront pas être appelés directement par un poste extérieur dans une application de visiophonie, par exemple.

Avec le développement de l'Internet à la maison, de nombreux objets communiquants devraient apparaître. Du nounours à l'extincteur qui demande à être révisé, ces objets embarqueront des composants qui doivent être simples et peu coûteux. Il devient donc difficile d'intégrer (et de configurer) des doubles piles dans ces "systèmes enfouis". L'Internet à la maison est un des grands domaines qui pousse à IPv6 pour que chaque objet puisse avoir sa propre adresse et ainsi être appelé. Cependant, si le décollage des objets connectés se fait avec le protocole IPv4, alors la migration sera très lente car il sera difficile de disposer d'objets ayant une double pile IPv4/IPv6.

Le choix parmi les différents scénarios de transition dépendra beaucoup de l'importance du manque d'adresses IPv4

6. Connecter des pilots IPv6 entre eux

Pour faire communiquer des réseaux IPv6 éloignés en passant par le monde IPv4, on utilise des "tunnels" où des paquets IPv6 sont encapsulés dans des paquets IPv4 et MPLS pour traverser les parties IPv4 du réseau.

*I*l existe deux principales méthodes de "tunnelling" :

- ✓ Le RFC 2893 propose un mécanisme de base limité (Cette solution permet au 6bone de communiquer avec l'extérieur)
- ✓ 6to4 : Une nouvelle technique permettant à des sites IPv6 isolés de se relier à d'autres en utilisant des routeurs "6to4". Ce protocole pourrait être utilisé par les services d'accès IPv6 en natif.

Il existe également des "tunnel brokers" qui agissent comme des fournisseurs de tunnels IPv6 offrant une connectivité Internet IPv6 mais à travers des tunnels permettant de relier un réseau IPv6 à d'autres réseaux IPv6 en passant par le "monde IPv4". Il en existe deux aujourd'hui, un au Canada et un en Europe. Leur nombre devrait rapidement se multiplier.

7. La stratégie de migration des fournisseurs d'accès

Les fournisseurs d'accès qui souhaitent migrer vers IPv6 devront faire cohabiter les deux mondes. Ils ont trois choix à leur disposition :

1. Faire du "tunnelling" pour permettre à votre machine IPv6 d'accéder au reste du monde IPv6 tout en conservant son réseau en IPv4 (probablement par la méthode 6to4)
2. Avoir deux infrastructures séparées : Une nouvelle pour IPv6 tout en gardant l'ancienne infrastructure IPv4
3. Avoir des serveurs disposant d'une double pile IPv4/IPv6

8. Quel est le plus complexe : passer à IPv6 ou garder IPv4 ?

Aujourd'hui, faire migrer un réseau existant d'IPv4 vers IPv6 représente un effort significatif. Modifier une configuration qui marche est toujours un risque. Le problème est moindre pour les nouveaux réseaux (mobiles, réseaux d'objets communiquants...), les problèmes techniques semblent bien maîtrisés aujourd'hui. Cependant, les administrateurs doivent apprendre à configurer des piles IPv6.

Progressivement, la complexité de conserver IPv4 augmente. La demande augmente pour des connections permanentes et des services nouveaux tels que la qualité de service. La configuration des adresses est plus complexe à mettre en oeuvre qu'avec IPv6. Au fur et à mesure que des réseaux IPv6 sont déployés, que les nouveaux produits intègrent IPv6 et que les compétences d'administration IPv6 se banalisent, il devient plus simple d'avoir un réseau IPv6 qu'un réseau IPv4.